UEE1303(1009) S22
Final Examination

FULL SCORES:

120 %

EXAMINATION TIME:

6/13 18：30 ～ 6/16 23：55

INSTRUCTIONS:

This final examination is composed of 3 different problem sets. You are allowed to open any notes or books, but prohibited to browse on the internet to search C/C++ codes as direct answers. Read carefully the statements and requirements of each problem. Please note that points will be given until your program fully fulfills the requirements of each problem. No partial credits will be given.

You should be aware of that some lengthy problems are not as difficult as you think. In contrast, some short problems may not be solved easily. Last but not least, **cheating** is a serious academic demeanor and should be avoided at all most. Once any cheating is caught, all your answers will be void and get 0 points for your midterm.

Good luck!

UEE1303(1009) S22: Final Examination Demo Sheet

Student ID # : ______________________, Name :_________________________

Full Scores: up to 120 points

➢ You may pick arbitrary numbers of problems to solve.

Problem Set 1 (40%)		Problem Set 2 (60%)	
Subproblem	**TA Signature**	**Subproblem**	**TA Signature**
Q1 (40%)		Q1 (40%)	
		Q2 (20%)	

Problem Set 3 (60%)			
Subproblem	**TA Signature**		
Q1 (20%)			
Q2 (40%)			

Total Score: ______________ / 120

【SOURCE CODE AND TEST CASE】

The path of the open test cases for each problem set:
`<PATH> /tmp/OOP_Final/p<problem index>`
Copy the all open test cases to your local directory and start to complete each problem in the corresponding folder.
`<EX> $ cp -R /tmp/OOP_Final ~/.`

【PROBLEM SET 1】(40%)

This problem is similar to the lab problem in that you must complete the file `P1.cpp` without adding any additional functions and ensure that the class structure matches the hierarchy figure by adding inheritance relations between classes.

You can add `virtual` to the functions or classes if you want. However, you cannot add any new functions or data members to this problem.

```
// P1.cpp PART1-Template

template <class T>
T *new_pvec(int n)
{
    /*
        FUNCTION:
        New the number of n spaces in type T.
    */
    // NEED SOME CODES
}

template <class T>
void copy_pvec(T *p1, T *p2, int n)
{
    /*
        FUNCTION:
        Copy the number of n spaces in type T from p2 to p1.
        You don’t need to allocate new spaces to p1.
    */
    // NEED SOME CODES
}

template <class T>
void del_pvec(T *vecs)
{
    /*
        FUNCTION:
        Delete an array.
    */
    // NEED SOME CODES
}

template <class T>
void display_pvec(T *vecs, int n)
{
    /*
        FUNCTION:
        Display the number of n elements using cout.
    */
    // NEED SOME CODES
}
```

```
// P1.cpp PART2-Point-Class
class Point2D
{
private:
    int *x;
    int *y;

public:
    Point2D(int n1, int n2)
    {
        /*
            FUNCTION:
            Constructor assigns memory and value.
            n1 -> x, n2 -> y.
        */
        // NEED SOME CODES
    }
    Point2D()
    {
        /*
            FUNCTION:
            Constructor assigns memory and random value in range [0,
999].
        */
        // NEED SOME CODES
    }
    Point2D(Point2D &p)
    {
        /*
            FUNCTION:
            Constructor copys data value from p.
        */
        // NEED SOME CODES
    }
    int getX()
    {
        return *x;
    }
    int getY()
    {
        return *y;
    }

    friend ostream &operator<<(ostream &out, Point2D &p);
    void operator=(Point2D &p)
    {
        /*
            FUNCTION:
            Assign values to the data members from p.
        */
        // NEED SOME CODES
    };
    void display() { cout << (*this); }
};

ostream &operator<<(ostream &out, Point2D &p)
{
    /*
        FUNCTION:
        As shown in the output file, display point p.
    */
    // NEED SOME CODES
}

class Point4D
{
private:
    int *z;
    int *t;

public:
    Point4D() : Point2D()
    {
        /*
            FUNCTION:
            Constructor assigns memory and random value in range [0,
999].
        */
        // NEED SOME CODES

    }
    ~Point4D()
    {
        /*
            FUNCTION:
            Delete memory of data members.
        */
        // NEED SOME CODES
    }
    void display() { cout << (*this); }
    friend ostream &operator<<(ostream &out, Point4D &p);
};
ostream &operator<<(ostream &out, Point4D &p)
{
    /*
        FUNCTION:
        As shown in the output, display point p.
    */
    // NEED SOME CODES
}
```

```
// P1.cpp PART3-Shape-Class
class Shape
{
protected:
    int color;
    Point2D *points;

public:
    void area() = 0;
    bool is_polygon() = 0;
    ~Shape() {}
};

class Polygon
{
public:
    bool is_polygon() { return true; }
};

class Circle
{
private:
    Point2D center;
    double radius;

public:
    Circle(Point2D p, double r) : center(p), radius(r) {}
    void area()
    {
        /*
            FUNCTION:
            Print the area as the output shows.
        */
        // NEED SOME CODES
    };
    double get_area(){
        /*
            FUNCTION:
            Return the area of this circle.
        */
        // NEED SOME CODES
    }
    bool is_polygon() { return false; }
};

class Triangle
{
private:
    Point2D *pvecs;

public:
    Triangle(Point2D *vecs)
    {
        pvecs = new Point2D[3];
        copy_pvec<Point2D>(pvecs, vecs, 3);
    }
    ~Triangle() { delete[] pvecs; }
    void area()
    {
        /*
            FUNCTION:
            Print the area as the output shows.
        */
        // NEED SOME CODES
    };
    double get_area(){
        /*
            FUNCTION:
            Return the area of this triangle.
        */
        // NEED SOME CODES
    }
};

class Triangle_and_Circle
{
public:
    Triangle_and_Circle(Point2D p, double r, Point2D *vecs) :
Triangle(vecs), Circle(p, r) {}
    void area()
    {
        /*
```

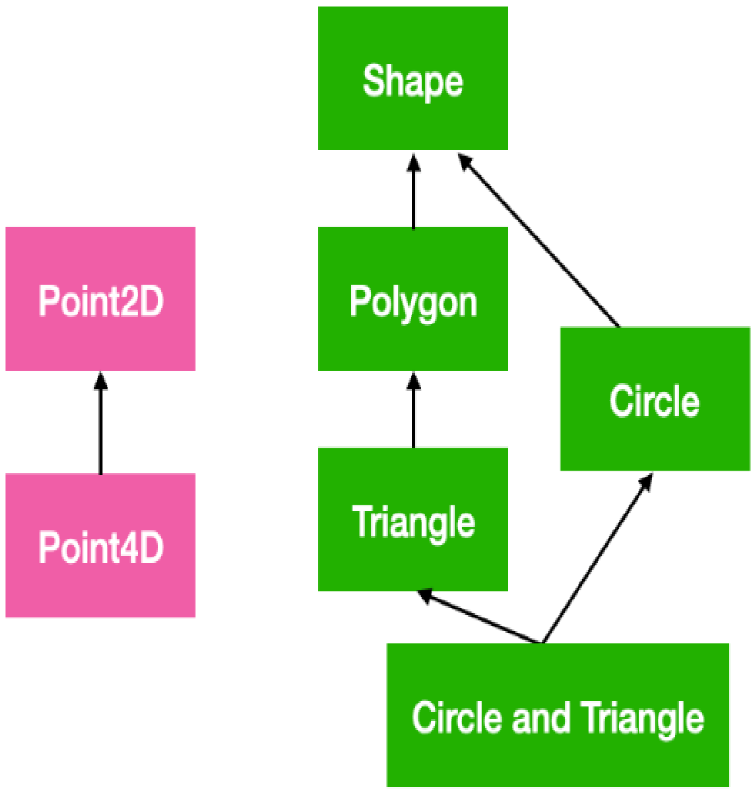
```
            FUNCTION:
            Print the area as the output shows.
        */
        // NEED SOME CODES
    };
    double get_area(){
        /*
            FUNCTION:
            Return the area of this triangle minus circle.
        */
        // NEED SOME CODES
    }

    bool is_polygon() { return false; }
};
```

```
// P1.cpp PART4-main

int main()
{
    srand(1);

    cout << "===== int vecs template =====" << endl;
    int *vec_int = new_pvec<int>(3);
    display_pvec(vec_int, 3);
    del_pvec(vec_int);
    vec_int = new int[4];
    vec_int[0] = 10;
    vec_int[1] = 20;
    vec_int[2] = 30;
    vec_int[3] = 40;
    int *vec_int2 = new int[4];
    copy_pvec(vec_int2, vec_int, 4);
    display_pvec(vec_int, 3);
    del_pvec(vec_int);
    del_pvec(vec_int2);
    cout << endl;


    cout << "===== Point2D p1 =====" << endl;
    Point2D p1(0, 0);
    cout << p1 << endl;
    cout << endl;


    cout << "===== Point2D p2 =====" << endl;
    Point2D p2;
    cout << p2 << endl;
    cout << endl;
    cout << "===== Point4D p4 =====" << endl;
    Point2D *p = new Point4D;
    p->display();
    cout << endl;


    cout << "===== new vec =====" << endl;
    Point2D *vec = new_pvec<Point2D>(3);
    display_pvec(vec, 3);
    cout << endl;


    cout << "===== Circle and Triangle =====" << endl;
    Triangle_and_Circle tc(p1, 10, vec);
    tc.area();
    if (tc.is_polygon())
        cout << "This is a polygon" << endl;
    else
        cout << "This is not a polygon" << endl;
    cout << endl;


    cout << "===== Delete =====" << endl;
    del_pvec<Point2D>(vec);
    cout << endl;


    cout << "===== Area of the Circle =====" << endl;
    tc.Circle::area();
    if (tc.Circle::is_polygon())
        cout << "This is a polygon" << endl;
    else
        cout << "This is not a polygon" << endl;
    cout << endl;


    cout << "===== Area of the Triangle =====" << endl;
    tc.Triangle::area();
    if (tc.Triangle::is_polygon())
        cout << "This is a polygon" << endl;
    else
        cout << "This is not a polygon" << endl;
    cout << endl;
    return 0;
}
```

Output:

```
$ g++ P1.cpp -o P1
$ ./P1
===== int vecs template =====
0
0
0
10
20
30

===== Point2D p1 =====
x=0 y=0

===== Point2D p2 =====
x=383 y=886

===== Point4D p4 =====
x=777 y=915 z=793 t=335
===== new vec =====
x=386 y=492
x=649 y=421
x=362 y=27

===== Circle and Triangle =====
61685.34073
This is not a polygon

===== Delete =====

===== Area of the Circle =====
Circle Area is 314.15927
This is not a polygon

===== Area of the Triangle =====
x=386 y=492
x=649 y=421
x=362 y=27
Triangle Area is 61999.50000
This is a polygon
```

Class Hierarchy



【PROBLEM SET 2】SHOP SIMULATOR (60%)

In this problem set, you are going to implement a shop simulator. In this simulator, the customers buy the groceries with a shopping basket in the shop. When a customer wants to check out, one needs to queue in line for a cashier to service him/her.

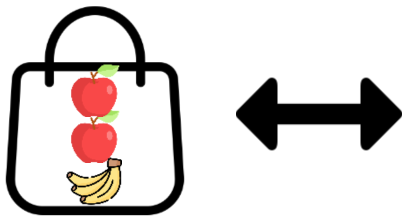
There are three types of customers which are men, women, and children. Each of these has three properties: name, gender, and age. When one's age is under 18 years old, he/she is considered a child. A man can keep five groceries in the shopping basket at most, a woman can keep four groceries in the shopping basket at most, and a child can only keep three groceries in the shopping basket at most.

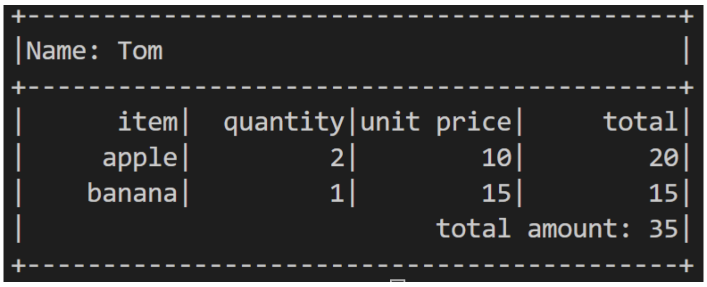
There are three kinds of groceries which are fruit, drink, and bread. Each of these has four properties: name, type, price, and weight. The behavior of a shopping basket is like a stack that is Last-In-First-Out. When a new grocery puts into the shopping basket, the basket has to maintain an order which is the bread needs to be on top of the fruit and the fruit needs to be on top of the drink. If the types of groceries are the same, the heavier one needs to be under the lighter one.

There are two kinds of cashiers. One is junior and the other is senior. A junior cashier can help two customers to check out at one time. A senior cashier can help three customers to check out at one time. When there is no free cashier, a customer needs to queue for check out.

A customer has three actions: entering the shop, buying something, and queuing for checkout. A cashier has three actions: opening a counter (starts to service customers), checking out for customers, and closing a counter (ends to service customers).

When a cashier checkouts for a customer, one will print out a receipt. A receipt contains each grocery's name, quantity, unit price, and total price. The top left corner of the receipt denotes the customer's name, and the down right corner of it denotes the total amount of the groceries. **The items are listed on the receipt in the order that they were removed from the basket**. An example of the mapping of a shopping basket to a receipt is shown as follows.





```
+----------------------------------------+
|Name: Tom                               |
+----------------------------------------+
|      item|  quantity|unit price|  total|
|     apple|         2|        10|     20|
|    banana|         1|        15|     15|
|                     total amount: 35|
+----------------------------------------+
```

The details of the receipt format are explained as follows. Each row contains **45** characters. Each type of grocery forms a row, and the row is composed of four columns. Each column of a grocery row is composed of **10** characters, separated each one by a verticle bar, and the context is right-aligned.

There are some rules in this shop.

- When a cashier finishes checking out for a customer, the customer will leave the shop.
- If there are some customers at the counter when the cashier closes the counter, they will go back to the **front** of the waiting queue. The first customer who arrives at the counter will become the first person in the waiting queue and so on.
- **If there are two and above counters that are free, a customer will go to the counter which is opened first.**

The scenario of the shop will be given as an input file, and you have to print some outputs to the terminal according to this file. Notice that you have to print a warning message under two situations. The first situation is when the customer wants to buy a grocery, but one's shopping basket can't take anymore. The second situation is when the cashier checks out for a customer who is not at the counter. You can suppose that no other unreasonable situations will happen except for these two situations. e.g., a customer won't go back shopping once one is waiting in the queue. The customer always has something to check out.

Besides, you have to implement a function to show the queueing situation and the customer(s) at the opening counter(s) when the input scenario file shows the text, `display`. The order of printing is, to print the queueing situation first and then print some counter(s) and the customer(s) at that counter. The earliest opened counter will be printed first and so on. If there is more than one customer in the queue, use an arrow to separate them. If there is more than one customer at the counter, use a space to separate them.

The following is an example. Showing that by giving two input files, `grocery.in` and `scenario.in`, what is the output showing on the terminal.

Input file 1: `grocery.in`

```
apple fruit 10 150
banana fruit 15 200
orange fruit 20 300
croissant bread 50 300
bagel bread 45 230
donut bread 40 250
coke drink 30 500
juice drink 50 400
coffee drink 40 300
```

Format of each row in `grocery.in`:

```
<1st item’s name> <1st item’s type> <1st item’s price> <1st item’s weight>
<2nd item’s name> <2nd item’s type> <2nd item’s price> <2nd item’s weight>
<3rd item’s name> <3rd item’s type> <3rd item’s price> <3rd item’s weight>
                                    …
```

Input file 2: `scenario.in`

```
enter Tom 20 M
enter Sam 7 M
buy Sam apple
buy Sam donut
buy Sam orange
buy Sam coke
enter Yuri 24 F
buy Tom croissant
queue Sam
display
open Cherry 20 F junior
buy Yuri coffee
queue Yuri
display
enter Henry 26 M
buy Henry bagel
queue Henry
display
checkout Cherry Henry
checkout Cherry Sam
display
enter Betty 18 F
buy Betty juice
queue Betty
display
close Cherry
display
enter Mike 5 M
buy Mike coke
buy Mike orange
buy Mike coke
queue Mike
open Amy 34 F senior
display
checkout Amy Henry
display
checkout Amy Jonathan
checkout Amy Mike
display
checkout Amy Yuri
checkout Amy Betty
```

```
display
close Amy
display
```

The possible actions appeared in `scenario.in` are listed as follows:

- A customer enters the shop

```
enter <customer’s name> <customer’s age> <customer’s gender>
```

- A customer buys a grocery

```
buy <customer’s name> <grocery’s name>
```

- A customer queues for checkout

```
queue <customer’s name>
```

- A cashier opens a counter

```
open <cashier’s name> <cashier’s age> <cashier’s gender> <cashier’s title>
```

- A cashier checks out for a customer

```
checkout <cashier’s name>  <customer’s name>
```

- A cashier closes a counter

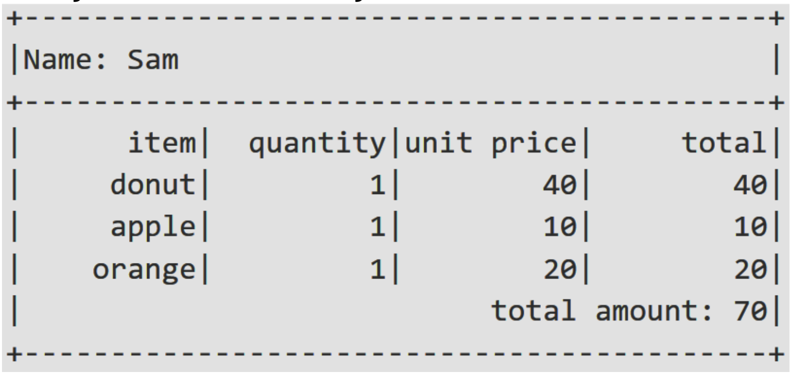
```
close <cashier’s name>
```

- Show the queuing situation

```
display
```

Output:

```
$ make
$ ./P2 grocery.in scenario.in
Sam can't buy anymore!
queue: Sam
queue:
Cherry's counter: Sam Yuri
queue: Henry
Cherry's counter: Sam Yuri
Henry isn't at Cherry's counter!
```



```
+-------------------------------------------+
|Name: Sam                                  |
+-------------------------------------------+
|      item|  quantity|unit price|     total|
|     donut|         1|        40|        40|
|     apple|         1|        10|        10|
|    orange|         1|        20|        20|
|                           total amount: 70|
+-------------------------------------------+
```

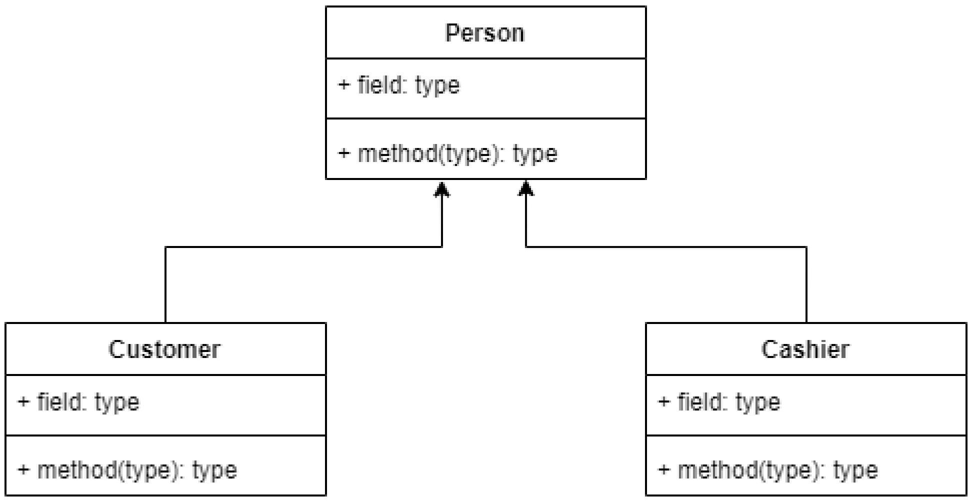
```
queue:
Cherry's counter: Yuri Henry
queue: Betty
Cherry's counter: Yuri Henry
queue: Yuri <- Henry <- Betty
queue: Mike
Amy's counter: Yuri Henry Betty
```

```
+----------------------------------------+
|Name: Henry                             |
+----------------------------------------+
|      item|   quantity|unit price|     total|
|     bagel|          1|        45|        45|
|                         total amount: 45|
+----------------------------------------+
queue:
Amy's counter: Yuri Betty Mike
Jonathan isn't at Amy's counter!
+----------------------------------------+
|Name: Mike                              |
+----------------------------------------+
|      item|   quantity|unit price|     total|
|    orange|          1|        20|        20|
|      coke|          2|        30|        60|
|                         total amount: 80|
+----------------------------------------+
queue:
Amy's counter: Yuri Betty
+----------------------------------------+
|Name: Yuri                              |
+----------------------------------------+
|      item|   quantity|unit price|     total|
|    coffee|          1|        40|        40|
|                         total amount: 40|
+----------------------------------------+
+----------------------------------------+
|Name: Betty                             |
+----------------------------------------+
|      item|   quantity|unit price|     total|
|     juice|          1|        50|        50|
|                         total amount: 50|
+----------------------------------------+
queue:
Amy's counter:
queue:
```

Each file (.h) should only contain a class. Please declare class header file (.h) and define its functionality in source file (.cpp).

Q1 (40%)
Your output should be correct.

Q2 (20%)
Your output should be correct, and you should use the inheritance hierarchy as below picture.



【PROBLEM SET 3】STRATEGY GAME (60%)

"Age of Empires" and "Sid Meier's Civilization" are currently well-known strategy games. This task requires you to program a 2D turn-based strategy(TDS) game that combines elements from two games.

In the TDS game, the objects of both players are stored on a 2D map. For instance, the size of map is (4,4), it indicates that

(0, 0)	(1, 0)	(2, 0)	(3, 0)
(0, 1)	(1, 1)	(2, 1)	(3, 1)
(0, 2)	(1, 2)	(2, 2)	(3, 2)
(0, 3)	(1, 3)	(2, 3)	(3, 3)

The origin (0, 0) is located in the upper left corner. Different colored cells represent different players. Each cell in the 2D map only has one object at most. According to the shared global counter between two players, each object has a unique identifier. The primary categories of objects are building and human. There are three types of "building" objects: house, farm or barracks. The description of the three objects is that

house object:

- 50 HP
- Generates 1 villager object per round
  - The new villager's position will be (X coordinate of the house, Y coordinate of the house + 1). If this cell is occupied by another object and the house is capable of creating a new villager, the house will pause the process of generating a villager until this cell is empty, at which point it will place a new villager in this cell.
- Creating this object needs 100 foods.

farm object:

- 100 HP
- generates 200 foods per round
- needs 1 villager objects
  - If the current total of villagers in this object is less than 1, the farm will reset the process of food production.
- If this object is destroyed, all villagers in it will be killed.
- Creating this object needs 200 foods.

barracks object:

- 300 HP
- 3 different types of barracks
  - Archer, Cavalry or spearman
  - Every two rounds, it generates 1 specific type of soldier and spends 300 foods.
    - If the current stock of food is less than 300, the barracks will pause the process of generating soldiers.

    - The new soldier's position will be (X coordinate of the barrack, Y coordinate of the barrack + 1). If this cell is occupied by another object and the barracks is capable of creating a new soldier, the barracks will pause the process of generating soldiers until this cell is empty, at which point it will place a new soldier in this cell.
- needs 2 villager objects
  - If the current total of villagers in this object is less than 2, the barracks will reset the process of generating soldiers.
- If this object is destroyed, all villagers in it will be killed.
- Creating this object needs 300 foods.

There are two distinct categories of "human" objects: villager and soldier. The description of the three objects is that

villager object:

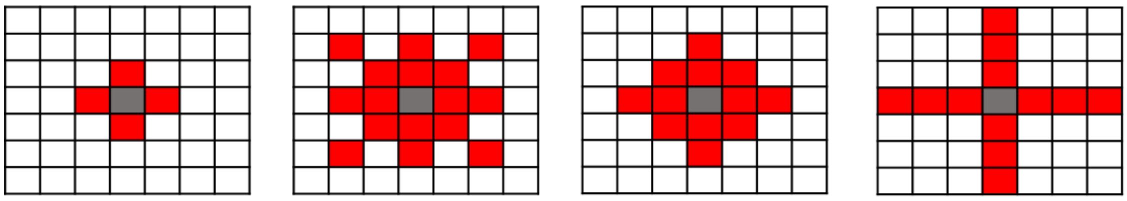
- 50 HP
- 1 speed point
- A villager could enter a farm or barracks in need of labor.
- If a villager object enters a building, it will disappear from a 2D map and be permanently recorded within the building.

soldier object:

- Different types of soldier information
  - Archer
    - 150 HP
    - 3 speed points
    - 30 base attack points
  - Cavalry
    - 300 HP
    - 2 speed points
    - 60 base attack points
  - Spearman
    - 500 HP
    - 4 speed points
    - 50 base attack points
- Type chart

Defending type / Attack type	Archer	Cavalry	Spearman	Building
Archer	1x	2x	1x	1x
Cavalry	1x	1x	2.5x	1x
Spearman	3x	0.5x	1x	1x

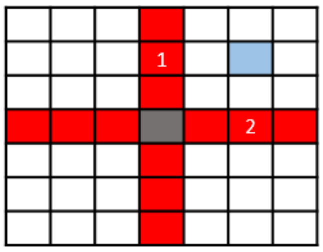
Moving range of a human object:



- red cell: reachable position, black cell: the position of this human object
- If there is an obstacle in this object's pathway, you do not worry that it will be obstructed.
- moving rule
  - target decision (first executed)
    - A villager will move automatically to the nearest farm or barracks in need of labor. If multiple farm/barrack objects have the same shortest *Euclidean distance*, this villager will approach the one with the smallest ID.
    - If no hostile object (human or building) is inside the soldier's attack range, he will move to the object that is the shortest Euclidean distance away. If many hostile objects have the same minimum Euclidean distance, this soldier will approach the one with the smallest ID.
  - position decision (second executed)
    - If no farm or barracks objects require labor, this villager will remain stationary.
    - If an enemy object (such as a human or building) is in this soldier's attack range or he is unable to move, he will remain immobile.
    - If multiple reachable positions in this round have the shortest *Euclidean distance* to the target, this human object will move to the location with the largest special value. The formula of this special value is that

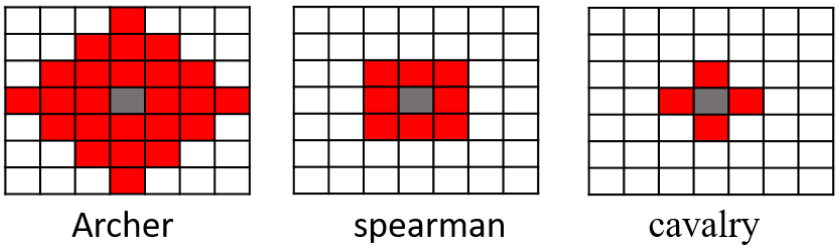
$$Diff.X = targetX - locationX$$
$$Diff.Y = targetY - locationY$$
$$F(x,y) = 2 * (Diff.X) * |Diff.X| + (Diff.Y) * |Diff.Y|$$

For example,



This image depicts how a cavalry soldier assaults the closest enemy target. According to the "target decision" policy, the blue cell is the target. There are two positions that have the shortest Euclidean distance to the target in this case. According to the formula, the cell containing the number 1 gets F(x,y) = 2*2*|2|+ 0*0 = 8, and the cell containing the number 2 gets F(x,y) = 2*0*0 + 2*|2| = 4. This soldier will then move to the cell containing number 1 because this cell has the highest specific value.

Attack range of a soldier object:



- red cell: attackable position, block cell: the position of this human object
- If a soldier object attacks another one, it must first perform a move action.
- If many hostile objects (human or building) are within this soldier's attack range, he will attack the one that is closest to him according to Euclidean distance. If many hostile objects have the same minimal Euclidean distance, this soldier will attack the one with the smallest ID.
- If there isn't any object in the attack range, the soldier object will skip the attack action.

In the beginning of the game, both players will have 1000 foods. Players could operate objects in each turn. Each turn consists of two distinct phases: "human object time" and "building object time," with the former executed before the latter. The description of the two periods is that

human object time:

- Only operate human objects
- Computer automatically manipulates objects in this stage.
- The operating objects are ordered by the speed of the object from largest to smallest. The object with the highest speed is activated first. If many human objects have the same speed, they will be operated from smallest to largest according to their ID.

building object time:

- Only operate building objects
- Players only manipulate objects in this stage.
- This stage is divided into two parts:" command time" and "operating time", the former will be executed before the latter.
    - command time
        - Player 1's commands will be executed before Player 2.
        - Each line stands for a command.

        - If he has sufficient resources, a player may execute numerous commands during each round. If this player is unable to execute this command, the system will transfer the control to the other player and this command will become this player's first command in the next round.
        - In each round, if a player has completed all of his commands, the system will bypass him and transfer control to the other player in the command time.
    - operating time
        - The operating objects are arranged from smallest to largest by their IDs. In this 2D map, for instance, the building object with the smallest ID will be executed first to generate food (farm obj.), a villager (house obj.), or a soldier (barrack obj.).

There are two input files:

p1_cmd_list.txt: command set of player1

p2_cmd_list.txt: command set of player2

- **Format of each input file**

```
< 1st object type> <1st position X> <1st position Y>
< 2nd object type> <2nd position X> <2nd position Y>
< 3rd object type> <3rd position X> <3rd position Y>
                              …
```

- Object type: 5 types
    - house
    - farm
    - archer_barracks
    - cavalry_barracks
    - spearman_barracks
- Position X: integer value (range: 0~100)
- Position Y: integer value (range: 0~100)

There is one output file:

game.log: record all operations of all rounds

- **Format of the output file**

```
Round 1:
 < operation of human object time in round 1>
 < operation of building object time in round 1>
Round 2:
 < format of human object time in round 2>
 < format of building object time in round 2>
                              …
```

- Operation of human object
    - 3 types of format
        - Move a human object

```
<human ID> moves to (<position X>, < position Y>)
```

        - Attack a human object

```
<attacker ID> (<position X of attacker>,<position Y
of attacker> ) attacks <defender ID> (<position X of
defender >,<position Y of defender > )
```

        - Destroy a human/building object

```
<attacker ID> destroys <defender ID>
```

            - You also need to print out "Attack operation" before "Destroy operation"
- Operation of building object
    - 2 types of format
        - Create new building

```
<new building ID> is at (<position X>, <position Y>)
```

        - Generate something

```
<building ID> generates <number of objects> <object type>
```

            - Object type: 3 types
                - foods
                - villager
                - soldier

Compile:

```
$ make
```

Execute:

```
$ ./P3 <path of p1_cmd_list.txt> <path of
p2_cmd_list.txt> <number of simulation rounds>
```

Test case:

```
$ make
$ ./P3 p1_cmd_list.txt p2_cmd_list.txt 12
Round 1:
ID_0 is at (5,5)
ID_1 is at (6,5)
ID_2 is at (7,5)
ID_3 is at (8,5)
ID_4 is at (9,6)
ID_5 is at (6,6)
ID_6 is at (20,20)
ID_7 is at (21,20)
ID_8 is at (24,21)
ID_9 is at (22,20)
```

```
ID_10 is at (23,20)
ID_11 is at (21,21)

Round 2:
ID_0 generates 1 villager
ID_2 generates 1 villager
ID_3 generates 1 villager
ID_6 generates 1 villager
ID_9 generates 1 villager
ID_10 generates 1 villager

Round 3:
ID_12 moves to (6,6)
ID_13 moves to (6,6)
ID_14 moves to (9,6)
ID_15 moves to (21,21)
ID_16 moves to (21,21)
ID_17 moves to (24,21)
ID_0 generates 1 villager
ID_2 generates 1 villager
ID_3 generates 1 villager
ID_6 generates 1 villager
ID_9 generates 1 villager
ID_10 generates 1 villager

Round 4:
ID_4 generates 200 foods
ID_8 generates 200 foods

Round 5:
ID_24 is at (5,7)
ID_25 is at (20,22)
ID_4 generates 200 foods
ID_8 generates 200 foods

Round 6:
ID_18 moves to (5,7)
ID_19 moves to (7,7)
ID_20 moves to (7,6)
ID_21 moves to (20,22)
ID_22 moves to (22,22)
ID_23 moves to (22,21)
ID_0 generates 1 villager
ID_3 generates 1 villager
ID_4 generates 200 foods
ID_5 generates 1 archer
ID_6 generates 1 villager
ID_8 generates 200 foods
ID_10 generates 1 villager
```

```
ID_11 generates 1 spearman

Round 7:
ID_31 moves to (19,22)
ID_28 moves to (8,9)
ID_19 moves to (6,7)
ID_20 moves to (7,7)
ID_22 moves to (21,22)
ID_23 moves to (22,22)
ID_26 moves to (5,7)
ID_29 moves to (20,22)
ID_0 generates 1 villager
ID_2 generates 1 villager
ID_4 generates 200 foods
ID_6 generates 1 villager
ID_8 generates 200 foods
ID_9 generates 1 villager

Round 8:
ID_31 moves to (19,20)
ID_28 moves to (10,11)
ID_4 generates 200 foods
ID_8 generates 200 foods

Round 9:
ID_31 moves to (18,19)
ID_28 moves to (12,13)
ID_4 generates 200 foods
ID_8 generates 200 foods
ID_24 generates 1 spearman
ID_25 generates 1 archer

Round 10:
ID_31 moves to (17,18)
ID_36 moves to (7,8)
ID_28 moves to (14,15)
ID_37 moves to (18,21)
ID_4 generates 200 foods
ID_8 generates 200 foods

Round 11:
ID_31 moves to (16,17)
ID_36 moves to (8,9)
ID_28 moves to (15,16)
ID_28 (15,16) attacks ID_31 (16,17)
ID_37 moves to (16,19)
ID_4 generates 200 foods
ID_8 generates 200 foods
ID_24 generates 1 spearman
```

```
ID_25 generates 1 archer

Round 12:
ID_31 (16,17) attacks ID_28 (15,16)
ID_31 destroys ID_28
ID_36 moves to (9,10)
ID_38 moves to (7,8)
ID_37 moves to (14,17)
ID_39 moves to (18,21)
ID_4 generates 200 foods
ID_8 generates 200 foods
```

Attention:

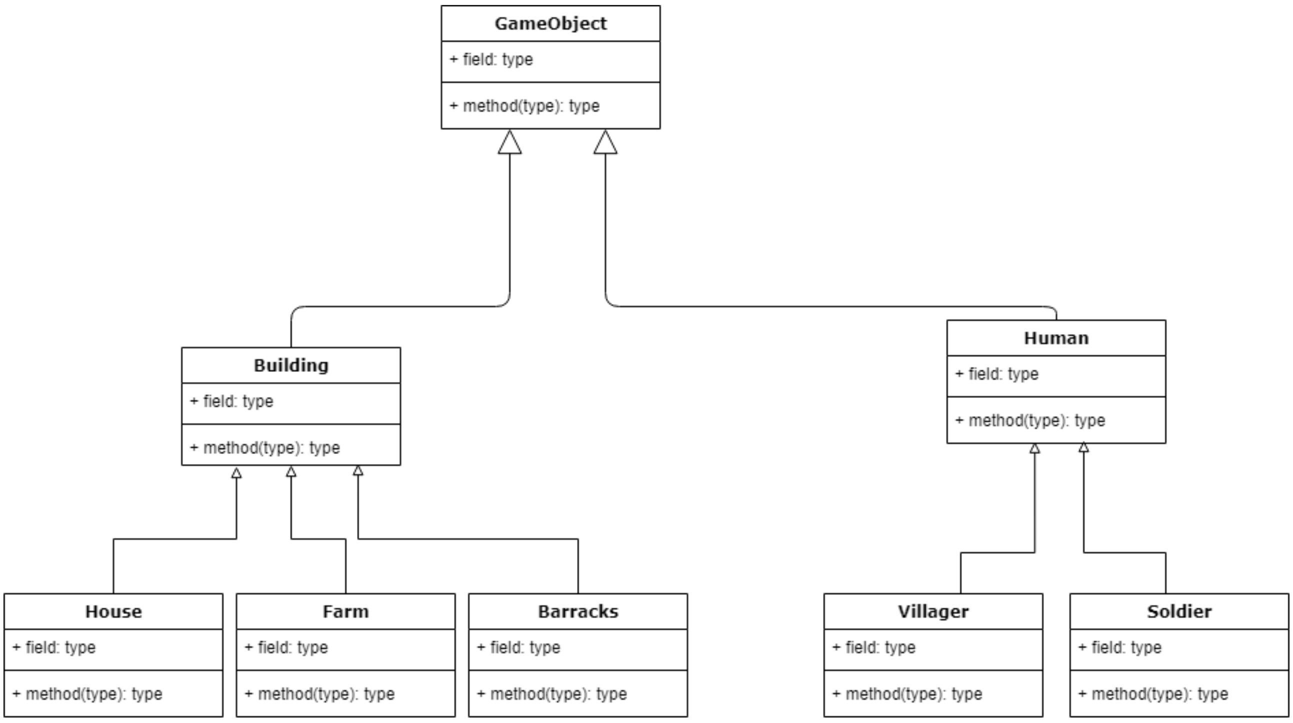
- Don’t care out of bounds problem
- Don't care how to end this game

Q1 (20%)

Your output should be correct.

Q2 (40%)

Your output should be correct. You should use multiple inheritance as in the picture below.


GameObject
+ field: type
+ method(type): type
Building
+ field: type
+ method(type): type
Human
+ field: type
+ method(type): type
House
+ field: type
+ method(type): type
Farm
+ field: type
+ method(type): type
Barracks
+ field: type
+ method(type): type
Villager
+ field: type
+ method(type): type
Soldier
+ field: type
+ method(type): type