COMP 2012 Final Exam - Fall 2017 - HKUST

Date: Dec 12, 2017 (Tuesday)
Time Allowed: 3 hours, 8:30 – 11:30 am
Instructions:

1. This is a closed-book, closed-notes examination.
2. There are **7** questions on **34** pages (including this cover page, an appendix, and 3 blank pages at the end). You may detach the appendix page and the blank sheets if you wish.
3. Write your answers in the space provided.
4. All programming codes in your answers must be written in the ANSI C++ version (i.e. version 2008) as taught in the class.
5. For programming questions, you are **NOT** allowed to define additional helper functions or structures, nor global variables unless otherwise stated. You also cannot use any library functions not mentioned in the questions. But you may use the STL functions given in the Appendix.

Student Name	
Student ID	
Email Address	
Lecture Section	L1 (TSOI, Desmond) / L2 (QUAN, Long)
Seat Number	

For T.A. Use Only

Problem	Topic	Score
1	True or False	/ 10
2	Inheritance and Dynamic Binding	/ 9
3	Binary Tree and Binary Search Tree (BST)	/ 8
4	Adelson-Velsky and Landis (AVL) Tree	/ 15
5	Hashing (Open Addressing)	/ 12
6	Binary Search Tree (BST)	/ 22
7	Abstract Base Class, Inheritance & Polymorphism	/ 24
Total		/ 100

Problem 1 [10 points] True or False

Indicate whether the following statements are *true* or *false* by circling **T** or **F**.
Each question carries 1 point.

T **F** (a) Static data members in any data types can be initialized in the class definition.

T **F** (b) Constructors and destructor cannot be inherited.

T **F** (c) A base class is constructed and destructed before the derived class.

T **F** (d) The following program compiles with error(s).

```
class Base {};
class Derived : protected Base {};
int main() {
  Base* b = new Derived;
  return 0;
}
```

T **F** (e) Once the destructor is declared virtual in the base class, the destructor in all directly or indirectly derived classes will automatically be made as virtual.

T **F** (f) Function overriding is only possible with inheritance.

T **F** (g) Abstract base class can be used as a function return type that is returned by value.

T **F** (h) A pure virtual function in an Abstract Base Class can be called from the constructor in the same class.

T **F** (i) It is NOT sufficient to construct the original Binary Search Tree if only the preorder traversal or the postorder traversal is given.

T **F** (j) Any subtree of an AVL tree is also an AVL tree.

Problem 2 [9 points] Inheritance and Dynamic Binding

```
#include <iostream>
using namespace std;

class Parent {
  private:
    int age;
  public:
    Parent() : age(50) { cout << "Parent default contor " << age << endl; }
    Parent(int age) { cout << "Parent conversion contor " << age << endl; }
    Parent(const Parent& p) {
      age = p.age; cout << "Parent copy contor " << age << endl;
    }
    void eat(string food) { cout << "Parent eats " << food << endl; }
};

class Child : public Parent {
  private:
    int age;
  public:
    Child() : age(25) { cout << "Child default contor " << age << endl; }
    Child(int age) { cout << "Child conversion contor " << age << endl; }
    Child(const Child& c) : age(c.age) {
      cout << "Child copy contor " << age << endl;
    }
    virtual void eat(string food) { cout << "Child eats " << food << endl; }
};

class GrandChild : public Child {
  private:
    int age;
  public:
    GrandChild(int age) { cout << "GrandChild " << age << endl;  }
    void eat(string food) { cout << "GrandChild eats " << food << endl; }
};

class GrandGrandChild : public GrandChild {
  private:
    int age;
  public:
    GrandGrandChild(int age) : GrandChild(age) {
      cout << "GrandGrandChild " << age << endl;
    }
    void eat(string food) { cout << "GrandGrandChild eats " << food << endl; }
};
```

```
int main() {
  GrandGrandChild ggc(5);

  cout << "------ End of block 1 -----" << endl;

  GrandChild gc(10);

  cout << "------ End of block 2 -----" << endl;

  Child c(gc);

  cout << "------ End of block 3 -----" << endl;

  Parent p(c);

  cout << "------ End of block 4 -----" << endl;

  Parent* pp = &c;
  pp->eat("Bread");
  pp = &gc;
  pp->eat("Hamburger");
  pp = &ggc;
  pp->eat("Candy");

  cout << "------ End of block 5 -----" << endl;

  Child* cp = &c;
  cp->eat("Congee");
  cp = &gc;
  cp->eat("Sandwiches");
  cp = &ggc;
  cp->eat("Biscuits");

  cout << "------ End of block 6 -----" << endl;

  GrandChild* gcp = &ggc;
  gcp->eat("Chocolate");

  cout << "------ End of block 7 -----" << endl;

  return 0;
}
```

What are the outputs of the above program? Write the outputs below.

Answer:

```
------ End of block 1 -----

------ End of block 2 -----

------ End of block 3 -----

------ End of block 4 -----

------ End of block 5 -----

------ End of block 6 -----

------ End of block 7 -----
```

Problem 3 [8 points] Binary Tree and Binary Search Tree (BST)

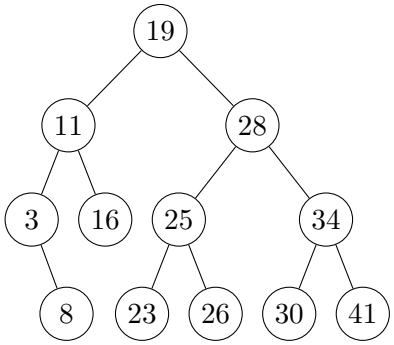
(a) [5 points] The preorder traversal of a binary tree is: A B C. Draw all the possible trees that yield this preorder traversal sequence.

Answer:

(b) [1 point] Identify the binary tree that you produced in part (a) yields the postorder traversal sequence: B C A.

Answer:

(c) [2 points] The following figure illustrates a Binary Search Tree (BST). We perform the following three operations on the tree, one after another:



i. Add 13

ii. Delete 19

iii. Delete 25

Draw the final BST after the three operations (the intermediate BST after each operation is not required).

Answer:

Problem 4 [15 points] Adelson-Velsky and Landis (AVL) Tree

Insert the following keys, in order (from left to right), into an initially empty AVL tree. Show the AVL tree after each insertion and re-balancing (if any).

```
18, 20, 72, 60, 79, 16, 14, 80, 75
```

Answer:

/*** Continue Your Answer For Problem 4 On This Page ***/

Problem 5 [12 points] Hashing (Open Addressing)

Fill in the following hash tables for each of the stated open addressing hashing methods when the following eight keys: 2, 3, 7, 33, 24, 29, 30, and 22 are inserted sequentially to the tables. The following hash function is to be used for this question:

$$hash(k) = k \mod 11$$

For each method, if the insertion of a key value fails after 11 probes, then we assume that the method fails. You may then stop, ignore all the remaining keys, and then provide the reason(s) of why the method fails. Note that, once a key value gets into the hash table, it stays in the table.

(a) [4 points] Linear probing: $h_i(k) = (hash(k) + i) \mod 11$

Table index	insert 2	insert 3	insert 7	insert 33	insert 24	insert 29	insert 30	insert 22
0								
1								
2								
3								
4								
5								
6								
7								
8								
9								
10								

(b) [4 points] Cubic probing: $h_i(k) = (hash(k) + i^3) \mod 11$

Table index	insert 2	insert 3	insert 7	insert 33	insert 24	insert 29	insert 30	insert 22
0								
1								
2								
3								
4								
5								
6								
7								
8								
9								
10								

(c) [4 points] Double hashing: Use the $hash_2$ function below as the second hash function:

$$hash_2(k) = 7 - (k \mod 7).$$

Table index	insert 2	insert 3	insert 7	insert 33	insert 24	insert 29	insert 30	insert 22
0								
1								
2								
3								
4								
5								
6								
7								
8								
9								
10								

Problem 6 [22 points] Binary Search Tree (BST)

Complete all the missing member functions of BST class according to the details given under Part(a)-(d) so that the class will work with the testing program "`BST-test.cpp`" and produce the given output.

```
#include "BST.h"

int main()      /* BST-test.cpp */
{
  BST<int> BSTree;

  cout << "Adding 15, 6, 18, 3, 7, 17, 20, 2, 4, 13, 9 and 14 to an empty BST";
  cout << endl << endl;

  BSTree.insert(15); BSTree.insert(6); BSTree.insert(18);
  BSTree.insert(3);  BSTree.insert(7); BSTree.insert(17);
  BSTree.insert(20); BSTree.insert(2); BSTree.insert(4);
  BSTree.insert(13); BSTree.insert(9); BSTree.insert(14);

  cout << "Resulting BST after insertion of keys:" << endl << endl;
  BSTree.printTree();
  cout << endl;

  cout << "All the keys between 6 and 18 (printed in descending order) are: " << endl;
  BSTree.print(6, 18);
  cout << endl << endl;

  cout << "No of internal nodes: " << BSTree.numInternal() << endl;
  cout << endl;

  int dist = BSTree.distBetweenTwoNodes(14, 15);
  cout << "Distance between 14 and 15 is " << dist << endl;

  dist = BSTree.distBetweenTwoNodes(7, 17);
  cout << "Distance between 7 and 17 is " << dist << endl;

  dist = BSTree.distBetweenTwoNodes(14, 18);
  cout << "Distance between 14 and 18 is " << dist << endl;

  dist = BSTree.distBetweenTwoNodes(2, 20);
  cout << "Distance between 2 and 20 is " << dist << endl;

  dist = BSTree.distBetweenTwoNodes(2, 14);
  cout << "Distance between 2 and 14 is " << dist << endl;

  dist = BSTree.distBetweenTwoNodes(17, 18);
  cout << "Distance between 17 and 18 is " << dist << endl;

  return 0;
}
```

Output of the testing program:

```
Adding 15, 6, 18, 3, 7, 17, 20, 2, 4, 13, 9 and 14 to an empty BST

Resulting BST after insertion of keys:

                20
        18
                17
 15
                                14
                        13
                                9
                7
        6
                        4
                3
                        2

All the keys between 6 and 18 (printed in descending order) are:
18, 17, 15, 14, 13, 9, 7, 6,

No of internal nodes: 6

Distance between 14 and 15 is 4
Distance between 7 and 17 is 4
Distance between 14 and 18 is 5
Distance between 2 and 20 is 5
Distance between 2 and 14 is 5
Distance between 17 and 18 is 1
```

```
#include <iostream>
#include <iomanip>          // For setw function
using namespace std;

template <typename T>       /* File: BST.h */
class BST {
  private:
    struct BSTNode {        // A node in a binary search tree
      T data;               // Node value
      BSTNode* left;        // Left child
      BSTNode* right;       // Right child
      BSTNode(const T& item, BSTNode* l = NULL, BSTNode* r = NULL)
          : data(item), left(l), right(r) {}
    };
    typedef BSTNode* BSTNodePointer;
    BSTNodePointer root;

    /***** Member functions that you need to implement *****/
    int numInternal(BSTNodePointer node) const;
    int distFromRoot(BSTNodePointer node, T v) const;
    int distBetweenTwoNodes(BSTNodePointer node, T x, T y) const;
    void print(BSTNodePointer node, T low, T high) const;

  public:
    BST() : root(NULL) {}   // Empty BST when its root is NULL
    ~BST() { delete root; }
    // Assume insert member function has been implemented.
    void insert(const T& item);
    int numInternal() const { return numInternal(root); }
    int distFromRoot(T v) const { return distFromRoot(root, v); }
    int distBetweenTwoNodes(T x, T y) const {
      return distBetweenTwoNodes(root, x, y);
    }
    void print(T low, T high) const { print(root, low, high); }
    // Assume printTree member function has been implemented.
    void printTree(int indent, BSTNodePointer node) const;
    void printTree() const { printTree(0, root); }
};

// BST-insert-printtree.tpp is given in the appendix,
// though it is not important for this question.
#include "BST-insert-printtree.tpp"
#include "BST.tpp"
```

```
/* File:  BST.tpp */

/* Part (a) [5 points]
 * Implement the member function:
 * int numInternal(BSTNodePointer node) const
 * Return the number of internal nodes (i.e. nodes that have child node(s)) in the BST
 * Return 0 if root is NULL or if there is only one node
 * You MUST use recursion.
 * ADD YOUR CODE BELOW
 */









/* Part (b) [4 points]
 * Implement the member function:
 * int distFromRoot(BSTNodePointer node, T v) const
 * Return the distance (i.e. the number of edges) from root to the node with value v
 * (For simplicity, you may assume that the value v always exists.)
 * You MUST use recursion.
 * ADD YOUR CODE HERE
 */
```

```
/* File:  BST.tpp */

/* Part (c) [7 points]
 * Implement the member function:
 * int distBetweenTwoNodes(BSTNodePointer node, T x, T y) const
 * Return the distance (i.e. the number of edges) between
 * the node with value x and the node with value y
 * (For simplicity, you may assume that the value x and y always exist.)
 * You MUST use recursion.
 * Hint: You may find distFromRoot member function useful.
 * ADD YOUR CODE BELOW
 */
```

```
/* File:  BST.tpp */

/* Part (d) [6 points]
 * Implement the member function:
 * void print(BSTNodePointer node, T low, T high) const
 * Output all the keys stored in the BST in descending order,
 * which satisfy low <= key <= high.
 * You MUST use recursion.
 * ADD YOUR CODE BELOW
 */
```

Problem 7 [24 points] Abstract Base Class, Inheritance & Polymorphism

This problem involves 5 classes called '`Position`', '`GameUnit`', '`Artillery`', '`Wizard`', and '`Troop`'. Below are the header files of the 5 classes.

```
#ifndef POSITION_H    /***** COMPLETED FOR YOU *****/
#define POSITION_H

#include <iostream>
using namespace std;

class Position {      /* File: Position.h */
  private:
    int x;  // The x-coordinate
    int y;  // The y-coordinate
  public:
    /* ----- Constructor: Construct a Position object with the given x, y ----- */
    Position(int x = 0, int y = 0) : x(x), y(y) {}

    /* ----- Accessor and mutator member functions ----- */
    int getX() const { return x; }
    int getY() const { return y; }
    void setX(int x) { this->x = x; }
    void setY(int y) { this->y = y; }

    /* ----- Distance function to compute the distance of two points ----- */
    double distance(const Position& p) {
      return (x - p.x) * (x - p.x) + (y - p.y) * (y - p.y);
    }

    /* ----- Overloading operator << ----- << */
    friend ostream& operator<<(ostream& os, const Position& p) {
      return os << "(" << p.x << ", " << p.y << ")";
    }
};

#endif /* POSITION_H */
```

```
#ifndef GAMEUNIT_H       /***** COMPLETED FOR YOU *****/
#define GAMEUNIT_H

#include <iostream>      /* File: GameUnit.h */
#include "Position.h"
using namespace std;

class GameUnit {
  private:
    string name;       // The name of the game unit
    double hp;         // The health point of the game unit
    int damage;        // The ability of the game unit to attack an enemy
    int defense;       // The ability of the game unit to withstand an attack
    Position position; // The current position of the game unit

  public:
    /* ----- Constructor and destructor ----- */

    // Construct a GameUnit with the given name, health point, damage, defense, and the
    // current position.
    GameUnit(string name = "", double hp = 0.0, int damage = 0,
             int defense = 0, Position position = Position())
             : name(name), hp(hp), damage(damage), defense(defense), position(position) { }

    // Virtual destructor of GameUnit
    virtual ~GameUnit() {}

    /* ----- Accessor and mutator member functions ----- */

    string getName() const { return name; }     // Return name of the GameUnit
    double getHP() const { return hp; }         // Return the health point of the GameUnit
    void setHP(double hp) { this->hp = hp; }    // Set the health point of the GameUnit
    int getDamage() const { return damage; }    // Return the damage of the GameUnit
    int getDefense() const { return defense; }  // Return the defense of the GameUnit
    // Return the current position of the GameUnit
    Position getPosition() const { return position; }
    // Set the position of the GameUnit
    void setPosition(Position position) { this->position = position; }


    /* ----- Pure virtual function ----- */

    // Attack the given enemy
    virtual void attack(GameUnit& enemy) = 0;

    // Print all data member values on screen
    virtual void print() const {
      cout << "Name: " << name << ", HP: " << hp << ", Damage: " << damage
           << ", Defense: " << defense << ", Position: " << position << endl;
    }
};

#endif /* GAMEUNIT_H */
```

```
#include "GameUnit.h"      /* File: Artillery.h */
using namespace std;

class Artillery : public GameUnit {
  private:
    double shootingRange;  // The shooting range of the artillery

  public:
    /* ----- Constructor ----- */

    // Construct an artillery with the given name, health point, damage, defense,
    // shootingRange, and the current position.

    Artillery(string name, double hp, int damage, int defense,
              double shootingRange, Position position)
              : GameUnit(name, hp, damage, defense, position), shootingRange(shootingRange)
    { }

    /* ----- Accessor and mutator member functions ----- */

    double getShootingRange() const { return shootingRange; }
    void setShootingRange(double shootingRange) { this->shootingRange = shootingRange; }

    /* ----- Attack the given enemy ----- */

    // - Check the distance between the artillery and the given enemy.
    //
    // - If the distance is within the shootingRange
    //   * Check if the damage of artillery (denoted by damage) is larger than the
    //     defense of the enemy (denoted by defense). If so, reduce the hp of the
    //     enemy by (damage - defense), and show the output statement as specified below:
    //     Attack: damage <enemy name>, value: <damage - defense>, HP remain: <enemy HP>
    //     (For details, refer to the provided sample output.)
    //   * Otherwise, check if the distance between the artillery and the given enemy is
    //     less than or equal to 1. If so, the artillery will be counterattacked and the
    //     hp of the artillery will be reduced by (defense - damage).
    //     Show the output statement as specified below:
    //     Counterattack: damage <name of artillery>, value: <defense - damage>,
    //     HP remain: <HP of artillery>
    //   * Otherwise, output "No attack nor counterattack"
    // - Otherwise, output "Out of attack range!"

    virtual void attack(GameUnit& enemy);   /***** IMPLEMENT THIS *****/

    // Display all the data member values of Artillery

    virtual void print() const;             /***** IMPLEMENT THIS *****/
};
```

```
#include "GameUnit.h"     /* File: Wizard.h */
using namespace std;

class Wizard : public GameUnit {
  private:
    double magicPower;    // The current magicPower of the Wizard

  public:
    /* ----- Constructor ----- */

    // Construct a wizard with the given name, health point, damage, defense,
    // magicPower, and the current position.

    Wizard(string name, double hp, int damage, int defense,
           double magicPower, Position position)
           : GameUnit(name, hp, damage, defense, position), magicPower(magicPower) { }

    /* ----- Accessor and mutator member functions ----- */

    double getMagicPower() const { return magicPower; }
    void setMagicPower(double magicPower) { this->magicPower = magicPower; }

    /* ----- Attack the given enemy ----- */

    // - Check the distance between the wizard and the given enemy.
    //
    // - If the distance is less than or equal to 1
    //   * Check if the damage of wizard (denoted by damage) is larger than the
    //     defense of the enemy (denoted by defense). If so, reduce the hp of the
    //     enemy by (damage - defense), and show the output statement as specified below:
    //     Attack: damage <enemy name>, value: <damage - defense>, HP remain: <enemy HP>
    //     (For details, refer to the provided sample output.)
    //   * Otherwise, the wizard will be counterattacked and the hp of the wizard will be
    //     reduced by (defense - damage). Show the output statement as specified below:
    //     Counterattack: damage <name of wizard>, value: <defense - damage>,
    //     HP remain: <HP of wizard>
    //     (For details, refer to the provided sample output.)
    //  - Otherwise, output "Out of attack range!"

    virtual void attack(GameUnit& enemy);     /***** IMPLEMENT THIS *****/

    // Display all the data member values of Wizard

    virtual void print() const;               /***** IMPLEMENT THIS *****/

    /* ----- Heal the wizard ----- */

    // - Check whether magicPower is greater than or equal to 10. If so, increase the HP
    //   of wizard by 10 and reduce magicPower by 10. Show the output statement as
    //   specified below:
    //   Heal: <name of wizard>, HP remain: <HP of wizard>,
    //   Magic power remain: <magicPower of wizard>
    //   (For details, refer to the provided sample output.)
    // - Otherwise, output "Not enough magic power!"

    void heal();                              /***** IMPLEMENT THIS *****/
};
```

```
#include <vector>      /* File: Troop.h */
#include <typeinfo>
#include "Artillery.h"
#include "Wizard.h"
#include "GameUnit.h"
using namespace std;

class Troop {
  private:
    // A vector container that contains a collection of GameUnit pointers
    vector<GameUnit*> gameUnits;

  public:
    // Default constructor
    Troop() { }

    /* ----- Copy constructor - Perform deep copying ----- */

    // Note:
    // - Two different types of objects will be pointed by the vector container.
    //   Create Artillery object when the object to be duplicated is in Artillery type.
    //   Create Wizard object when the object to be duplicated is in Wizard type.
    // Hint:
    // - typeid(<type>) will return a typeinfo object that carries the type information.
    //   typeid(<object>) will return a typeinfo object that carries the type of <object>.
    //   operator== has been overloaded for typeinfo class.
    Troop(const Troop& troop);             /***** IMPLEMENT THIS *****/

    /* ----- Destructor ----- */

    // De-allocate all the dynamically-allocated memory to avoid any memory leak as
    // the program finishes.
    ~Troop();                              /***** IMPLEMENT THIS *****/

    /* ----- Add game unit ----- */

    // Adds gameUnit to the back of the vector container
    void addGameUnit(GameUnit* gameUnit);  /***** IMPLEMENT THIS *****/

    /* ----- Overload operator[] for Troop ----- */

    // Returns the pointer at index n in the container.
    // If index n is out of legal range, return NULL
    GameUnit* operator[](int n);           /***** IMPLEMENT THIS *****/
};
```

Below is the testing program "GameEngine.cpp"

```
#include <string>
#include "Troop.h"
using namespace std;

int main() {
  Troop troopA;
  GameUnit* gameUnit = new Artillery("Jason", 150.5, 60, 30, 10, Position(3,5));
  troopA.addGameUnit(gameUnit);
  gameUnit = new Wizard("Peter", 100.5, 50, 100, 100.5, Position(1,2));
  troopA.addGameUnit(gameUnit);

  Troop troopB;
  gameUnit = new Artillery("Lawrence", 130.5, 70, 40, 10, Position(2,2));
  troopB.addGameUnit(gameUnit);
  gameUnit = new Wizard("Coolman", 90.5, 30, 20, 20.5, Position(3,4));
  troopB.addGameUnit(gameUnit);

  cout << "Troop A: Jason (Artillery) attacks Troop B: Coolman (Wizard)" << endl;
  cout << "-----------------------------------------------------------" << endl;
  troopA[0]->print();
  troopB[1]->print();
  troopA[0]->attack(*troopB[1]);
  if(typeid(*troopB[1]) == typeid(Wizard))
    ((Wizard*)troopB[1])->heal();

  cout << endl << endl;

  cout << "Troop B: Lawrence (Artillery) attacks Troop A: Peter (Wizard)" << endl;
  cout << "-----------------------------------------------------------" << endl;
  troopB[0]->print();
  troopA[1]->print();
  troopB[0]->attack(*troopA[1]);

  cout << endl << endl;

  cout << "Troop B: Lawrence (Artillery) moves to (3,3) and attacks "
       << "Troop A: Peter (Wizard)" << endl;
  cout << "-----------------------------------------------------------"
       << "----------------------" << endl;
  troopB[0]->setPosition(Position(3,3));
  troopB[0]->print();
  troopA[1]->print();
  troopB[0]->attack(*troopA[1]);

  return 0;
}
```

A sample run of the test program is given as follows:

```
Troop A: Jason (Artillery) attacks Troop B: Coolman (Wizard)
------------------------------------------------------------
Name: Jason, HP: 150.5, Damage: 60, Defense: 30, Position: (3, 5)
Shooting range: 10
Name: Coolman, HP: 90.5, Damage: 30, Defense: 20, Position: (3, 4)
Magic power: 20.5
Attack: damage Coolman, value: 40, HP remain: 50.5
Heal: Coolman, HP remain: 60.5, Magic power remain: 10.5


Troop B: Lawrence (Artillery) attacks Troop A: Peter (Wizard)
------------------------------------------------------------
Name: Lawrence, HP: 130.5, Damage: 70, Defense: 40, Position: (2, 2)
Shooting range: 10
Name: Peter, HP: 100.5, Damage: 50, Defense: 100, Position: (1, 2)
Magic power: 100.5
Counterattack: damage Lawrence, value: 30, HP remain: 100.5


Troop B: Lawrence (Artillery) moves to (3,3) and attacks Troop A: Peter (Wizard)
------------------------------------------------------------------------------
Name: Lawrence, HP: 100.5, Damage: 70, Defense: 40, Position: (3, 3)
Shooting range: 10
Name: Peter, HP: 100.5, Damage: 50, Defense: 100, Position: (1, 2)
Magic power: 100.5
No attack nor counterattack
```

Based on the given information, complete the missing member functions of '`Artillery`' class, '`Wizard`' class, and '`Troop`' class in their respective .cpp files, namely, "`Artillery.cpp`", "`Wizard.cpp`", and "`Troop.cpp`".

(a) [7 points] Implement the following member functions of the class '`Artillery`' in a separate file called "`Artillery.cpp`" according to the details given in "`Artillery.h`".

- `void attack(GameUnit& enemy);`
- `void print() const;`

Answer: `/* File "Artillery.cpp" */`

(b) [8 points] Implement the following member functions of the class '`Wizard`' in a separate file called "`Wizard.cpp`" according to the details given in "`Wizard.h`".

- `void attack(GameUnit& enemy);`
- `void print() const;`
- `void heal();`

Answer: `/* File "Wizard.cpp" */`

/*** Continue Your Answer For Problem 7(b) On This Page ***/

(c) [9 points] Implement the following member functions of the class '`Troop`' in a separate file called "`Troop.cpp`" according to the details given in "`Troop.h`".

- `Troop(const Troop& troop);`
- `~Troop();`
- `void addGameUnit(GameUnit* gameUnit);`
- `GameUnit* operator[](int n);`

Answer: `/* File "Troop.cpp" */`

/*** Continue Your Answer For Problem 7(c) On This Page ***/

---------- END OF PAPER ----------

Appendix

(1) STL Sequence Container: Vector

```
template <class T, class Alloc = allocator<T> > class vector;
```
Defined in the standard header **vector**.

Description:

Vectors are sequence containers representing arrays that can change in size. Just like arrays, vectors use contiguous storage locations for their elements, which means that their elements can also be accessed using offsets on regular pointers to its elements, and just as efficiently as in arrays. But unlike arrays, their size can change dynamically, with their storage being handled automatically by the container.

Some of the member functions of the vector<T> container class where T is the type of data stored in the vector are listed below.

Member function	**Description**
vector( )	Default constructor
iterator begin( ) const_iterator begin( ) const	Returns an iterator pointing to the first element in the vector. If the vector object is const-qualified, the function returns a const_iterator. Otherwise, it returns iterator.
iterator end( ) const_iterator end( ) const	Returns an iterator referring to the past-the-end element in the vector container. If the vector object is const-qualified, the function returns a const_iterator. Otherwise, it returns iterator.
void push_back(const T& val)	Adds a new element, val, at the end of the vector, after its current last element. The content of val is copied (or moved) to the new element.
T& operator[ ](int n) const T& operator[ ](int n) const	Returns a reference to the element at position n in the vector container.
int size() const	Returns the number of elements in the vector.

(2) **BST-insert-printtree.tpp**

```
template <typename T>
void BST<T>::insert(const T& item) {
  BSTNodePointer curPtr = root, parent = NULL;
  bool found = false;
  // First find the element
  while(!found && curPtr != NULL) {
    parent = curPtr;
    if(item < curPtr->data) // Descend left
      curPtr = curPtr->left;
    else if(curPtr->data < item) // Descend right
      curPtr = curPtr->right;
    else // Item found
      found = true;
  }
  if(!found) {
    curPtr = new BSTNode(item); // Construct node containing item
    if(parent == NULL) // Empty tree
      root = curPtr;
    else if(item < parent->data) // Insert to left of parent
      parent->left = curPtr;
    else // Insert to right of parent
      parent->right = curPtr;
  }
  else
    cout << "Item already in the tree" << endl;
}

template <typename T>
void BST<T>::printTree(int indent, BSTNodePointer node) const {
  if(!node)
    return;
  printTree(indent + 8, node->right);
  cout << setw(indent) << " " << node->data << endl;
  printTree(indent + 8, node->left);
}
```

/* Rough work */

/* Rough work */

/* Rough work */