.

COMP 2012 Final Exam - Fall 2020 - HKUST

Date:	December 18, 2020 (Friday)
Time Allowed:	2 hours, 4:30pm–6:50pm (with breaks)

Instructions:

1. This is a closed-book, closed-notes examination.
2. There are 6 questions in 3 separate parts A, B and C.
3. Type your answers in the space provided on Canvas.
4. All programming codes in your answers must be written in the ANSI C++ version as taught in the class.
5. For programming questions, unless otherwise stated, you are **NOT** allowed to define additional structures, classes, helper functions and use global variables, auto, nor any library functions not mentioned in the questions.

COMP 2012 Final Exam - Fall 2020 - HKUST: Part A

Time Allowed: 40 minutes

Instructions:

1. There are 4 short questions in this part.
2. This is a closed-book, closed-notes examination.
3. Type your answers in the space provided on Canvas.
4. All programming codes in your answers must be written in the ANSI C++ version as taught in the class.
5. For programming questions, unless otherwise stated, you are NOT allowed to define additional structures, classes, helper functions and use global variables, auto, nor any library functions not mentioned in the questions.

Problem	Topic	**Score**
1	Hashing	/ 10
2	AVL Tree	/ 10
3	Inheritance and Polymorphism	/ 10
4	Function Object and STL	/ 10
	Total	/ 40

Problem 1 [10 points] Hashing

(a)

0	1	2	3	4	5	6
yuqing	wallace	jasmine	arthur	desmond	EMPTY	hing

Total number of collisions occurred: $0 + 0 + 0 + 0 + 1 + 2 = 3$

(b)

0	1	2	3	4	5	6
EMPTY	wallace	jasmine	arthur	desmond	yuqing	hing

Total number of collisions occurred: $0 + 0 + 0 + 0 + 1 + 3 = 4$

Marking scheme:

- Each correct insertion gives 0.5 point. (7 points in total)
  If a student makes a mistake at one insertion but the next insertions are "consistent" with this mistake, then they should only lose 0.5 point for the original mistake.
- Each correct number of collisions gives 1.5 points. (3 points in total)

Problem 2 [10 points] AVL Tree

Level order traversal of the AVL tree that results from successive key insertions / deletion
31, 20, 42
31, 18, 42, 15, 20
31, 18, 45, 15, 20, 42, 50
31, 18, 45, 15, 27, 42, 50, 20, 29
31, 18, 45, 15, 27, 36, 50, 20, 29, 33, 42
31, 18, 36, 15, 27, 33, 50, 20, 29, 42

Marking scheme:

For each line of output

- Give 2 points if the output sequence is exactly the same as the solution.
- Give 1 point if the output sequence is not the same, but it forms an AVL tree (use the given program to check whether the sequence forms an AVL tree).
  Note: Putting arbitrary or irrelevant values is not accepted.

Problem 3 [10 points] **Inheritance and Polymorphism**

```
*** Block 1 ***
Person ctor: Peter
Person ctor: John
Person copy ctor: John
Person ctor: Aaron
Doctor default ctor
Name: Aaron
Specializing field: Surgeon
Registration free rate: 3
Person dtor: Peter

*** Block 2 ***
Person ctor: Jack
Doctor default ctor
Name: Jack
Specializing field: Physician
Registration free rate: 4
Doctor dtor
Person dtor: Jack

*** Block 3 ***
Person ctor: Elvis
Patient default ctor
Name: Elvis
Social security Number: 0134-443

*** Block 4 ***
Person copy ctor: Elvis
Person copy ctor: Aaron
Bill default ctor
[ Doctor information ]
Name: Aaron
Specializing field: Surgeon
Registration free rate: 3
[ Patient information ]
Name: Elvis
Social security Number: 0134-443

*** Block 5 ***
Doctor dtor
Person dtor: Aaron
Patient dtor
Person dtor: Elvis
Patient dtor
Person dtor: Elvis
Doctor dtor
Person dtor: Aaron
Person dtor: John
Person dtor: John
```

Marking scheme:

- No points for the blank lines and no penalty for missing them.
- No points for the partition lines with ***, but there is a penalty for missing them: -0.25 point for each missing line.
- Simply put, 0.25 point for each meaningful ouptut line, except (1) the blank lines and partition lines.
- The outputs have to be given in the order of the concepts.
- Partial credits are given based on identifying the concepts from top to bottom as much as we can.
- Penalty for extra lines: -0.25 each.

Problem 4 [10 points] Function Object and STL

(a) [2 points]

```
friend class PersonSorter;          // 1 pint
friend void print(vector<Person>&); // 1 point
```

(b) [8 points]

```
class PersonSorter { // 2 points
  private:
    string field;    // 1 point
  public:
    PersonSorter(string field) : field(field) { }         // 1 point
    bool operator()(const Person& p1, const Person& p2) { // 2 points
      if(field == "name")          // 0.5 point
        return p1.name < p2.name;  // 0.5 point
      else if(field == "age")      // 0.5 point
        return p1.age < p2.age;    // 0.5 point
    }
};
```

COMP 2012 Final Exam - Fall 2020 - HKUST: Part B

Time Allowed: 40 minutes

Instructions:

1. There is 1 long question in this part.
2. This is a closed-book, closed-notes examination.
3. Type your answers in the space provided on Canvas.
4. All programming codes in your answers must be written in the ANSI C++ version as taught in the class.
5. For programming questions, unless otherwise stated, you are NOT allowed to define additional structures, classes, helper functions and use global variables, auto, nor any library functions not mentioned in the questions.

Problem	Topic	**Score**
5	Inheritance, Polymorphism and Dynamic Binding	/ 30

Problem 5 [30 points] Inheritance, Polymorphism and Dynamic Binding

(a)

```
/* File: Cat.h */

#ifndef CAT_H
#define CAT_H

#include "Pet.h"
#include <iostream>
using namespace std;

class Cat : public Pet { // 0.5 point
  public:                // 0.5 point
    Cat(string name)     // 0.5 point
      : Pet(name) { };   // 0.5 point for MIL; 0.5 for empty body

    // Destructor is optional

    void speak() const { // virtual keyword is optional but const is a must
                         // 0.5 point for "const", 0.5 point for the other
                         // part of the function header
      cout << "meow" << endl; // 0.5 point for "meow", 0.5 point for endl
    }
};

#endif
```

(b)

```
/* File: Dog.h */

#ifndef DOG_H
#define DOG_H

#include "Pet.h"
#include <iostream>
using namespace std;

class Dog : public Pet { // 0.5 point
  public:                // 0.5 point
    Dog(string name)     // 0.5 point
      : Pet(name) { };   // 0.5 point for MIL; 0.5 for empty body

    // Destructor is optional

    void speak() const { // virtual keyword is optional, but const is a must
                         // 0.5 point for the "const", 0.5 point for the
                         // other part of the function header
      cout << "woof" << endl; // 0.5 point for cout << "woof", 0.5 point for endl
    }
};

#endif
```

(c)

```
/* File: Petshop.cpp */

#include "Petshop.h"

PetShop::PetShop(string name) : name(name) { // Setting name: 0.5 point
  pets = nullptr;
  petCount = 0; // 0.5 point for setting both pets and petCount
} // Total point = 1

PetShop::PetShop(const PetShop& another) {
  petCount = 0;    // Necessary if assignment operator is to be used
  pets = nullptr;  // Necessary if assignment operator is to be used
                   // 0.5 point for both lines
  *this = another; // 1 point
} // Total points = 1.5

PetShop& PetShop::operator=(const PetShop& another) {
  if(&another != this) { // Avoid self-assignment // 0.5 point
    for(int i=0; i<petCount; i++)
      delete pets[i]; // 0.5 point
    delete [] pets;   // 0.5 point
    petCount = 0;     // 0.5 point

    name = another.name; // 0.5 point

    for(int i=0; i<another.petCount; i++) {          // 0.5 point
      if(typeid(*another.pets[i]) == typeid(Cat))    // 1 point
        addPet(new Cat(another.pets[i]->getName())); // 1 point
      else
        addPet(new Dog(another.pets[i]->getName())); // 1 point
    }
  }
  return *this; // 0.5 point
} // Total points = 6.5

PetShop::~PetShop() {
  for(int i=0; i<petCount; i++)
    delete pets[i]; // 0.5 point
  delete [] pets;   // 0.5 point
} // Total points = 1
```

```
bool PetShop::addPet(Pet* pet) {
  // 1 point for simply returning false if pet's name already exists
  for(int i=0; i<petCount; i++)
    if(pet->getName() == pets[i]->getName())
      return false;

  Pet** temp = new Pet*[petCount + 1]; // 0.5 point
  for(int i=0; i<petCount; i++)
    temp[i] = pets[i];  // 0.5 point
  temp[petCount] = pet; // 0.5 point
  delete [] pets;       // 0.5 point
  pets = temp;          // 0.5 point
  petCount++;           // 0.5 point

  return true; // 0.5 point
} // Total points = 4.5

bool PetShop::removePet(string name) {
  // 1 point for simply returning false if name does not exist

  for(int i=0; i<petCount; i++) {
    if(name == pets[i]->getName()) {     // 0.5 point
      Pet** temp = new Pet*[petCount-1]; // 0.5 point
      for(int j=0, k=0; k<petCount; j++, k++) {
        if(k==i) {
          delete pets[k];   // 0.5 point
          j--;
        }
        else
          temp[j] = pets[k]; // 0.5 point
      }
      delete [] pets; // 0.5 point
      pets = temp;    // 0.5 point
      petCount--;     // 0.5 point
      return true;    // 0.5 point
    }
  }

  return false;
} // Total points = 5
```

```
void PetShop::printPets() const {
  for(int i=0; i<petCount; i++) {
    cout << pets[i]->getName() << " the "; // 0.5 point
    if(typeid(*pets[i]) == typeid(Cat))    // 0.5 point
      cout << "Cat";
    else
      cout << "Dog";
    cout << " is in the shop!" << endl;
    // 0.5 point for all cout statments for "Cat" and "Dog" and " is in the shop! "
  }
} // Total points = 1.5
```

Note: Some common syntax error mark deduction may be put at Q10 comment box of Part B.

COMP 2012 Final Exam - Fall 2020 - HKUST: Part C

Time Allowed: 40 minutes

Instructions:

1. There is 1 long question in this part.
2. This is a closed-book, closed-notes examination.
3. Type your answers in the space provided on Canvas.
4. All programming codes in your answers must be written in the ANSI C++ version as taught in the class.
5. For programming questions, unless otherwise stated, you are NOT allowed to define additional structures, classes, helper functions and use global variables, auto, nor any library functions not mentioned in the questions.

Problem	Topic	**Score**
6	Binary Search Tree (BST)	/ 30

Problem 6 [30 points] Binary Search Tree (BST)

(a) [4.5 points]

```
template <typename T>    // 0.5 point for the prototype
void BST<T>::sort(const BST& bst, vector<BSTnode*>& nodes) const {
  if(!bst.is_empty()) {            // 0.5 point
    sort(bst.root->left, nodes);   // 1 point
    nodes.push_back(new BSTnode(bst.root->value)); // 1.5 points
    sort(bst.root->right, nodes);  // 1 point
  }
}
```

(b) [6 points]

```
template <typename T>   // 0.5 point for the prototype
void BST<T>::build_balanced_tree_helper(BST& bst, const vector<BSTnode*>& nodes,
                                        int start, int end) const {
  if(start <= end) {              // 0.5 point
    int mid = (start + end) / 2;  // 1 point
    bst.root = nodes[mid];        // 1 point
    build_balanced_tree_helper(bst.root->left, nodes, start, mid-1); // 1.5 points
    build_balanced_tree_helper(bst.root->right, nodes, mid+1, end);  // 1.5 points
  }
}
```

(c) [7.5 points]

```
template <typename T>  // 0.5 point for the prototype
const BST<T>& BST<T>::kth_smallest_helper(const BST& bst, int& k) const {
  if(bst.is_empty())   // 0.5 point
    return dummy;      // 0.5 point
  const BST& left_tree = kth_smallest_helper(bst.root->left, k); // 2 points
  if(left_tree.root != nullptr) // 0.5 point
    return left_tree;  // 0.5 point
  --k;                 // 0.5 point
  if(k == 0)           // 0.5 point
    return bst;        // 0.5 point

  return kth_smallest_helper(bst.root->right, k); // 1.5 points
}
```

(d) [12 points]

```
template <typename T>  // 0.5 point for the prototype
bool BST<T>::pair_add_up_to_value(int value) const {
  vector<BSTnode*> nodes;       // 0.5 point
  sort(*this, nodes);           // 1 point
  int start = 0;                // 0.5 point
  int end = nodes.size() - 1;   // 0.5 point
  while(start < end) {          // 1 point
    int start_value = nodes[start]->value;            // 0.5 point
    int end_value = nodes[end]->value;                // 0.5 point
    if(start_value + end_value == value) {            // 1 point
      cout << "Pair found: " << start_value << " + "
           << end_value << " = " << value << endl;    // 1.5 points
      return true;                                    // 0.5 point
    }
    if(start_value + end_value > value)               // 1 point
      --end;                                          // 0.5 point
    if(start_value + end_value < value)               // 1 point
      ++start;                                        // 0.5 point
  }
  cout << "No such pair found" << endl;               // 0.5 point
  return false;                                       // 0.5 point
}
```

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