

The Hong Kong University of Science and Technology

Department of Computer Science and Engineering

COMP2012: Object-Oriented Programming and Data Structures (Fall 2013)

Final Examination

Date: 17 Dec 2013 (Tue)

Time: 16:30 – 19:30 (3 hours)

- Your answers will be graded according to correctness, efficiency, precision, and clarity
- Mobile phones or any devices that can emit sound must be switched off
- This booklet has **NUMPAGE** single-sided pages. Please check that all pages are properly printed. You may use the reverse side of the pages for your rough work. If you decide to use the reverse side to continue your work, please clearly write down the question number

Student Name	Solution
Student ID	No photo taking Don't take away it
ITSC email	_____ @stu.ust.hk

Question	Score / Max. score
Q1. Stack and its application	/5
Q2. C++ Template	/5
Q3. Tree transversal and node deletion in BST	/10
Q4. Hashing	/10
Q5. Binary Search Tree	/10
Q6. Heapsort	/15
Q7. Generic Programming	/20
Q8. Polymorphism and Inheritance	/25
Total	/100

Q1. Stack and its application (5 marks)

Given the following class CharStack which is used to store a stack of characters:

```
// You can assume this class is already implemented
class CharStack {
public:
    CharStack(); // Constructor of an empty stack
    void push(char letter); // push an item to the top of stack
    char top(); // top of a stack, the stack must be non-empty
    void pop(); // remove an item from the top of the stack
    bool isEmpty(); // return true if a stack is empty
};
```

Given an input string, push digits (i.e. 0-9) to a stack and print them in a reverse order after all other letters have been printed out. For example

- *Input: ab432c1d*
- *Output: abcd1234*

```
int main() {
    string str;
    cin >> str;    // an input string
    CharStack s;   // an empty character stack

    // Write your code here
    for (int i = 0; i < str.size(); ++i)
    {
        if (str[i] >= '0' && str[i] <= '9')
            s.push(str[i]);
        else
            cout << str[i];
    }
    while(!s.isEmpty())
    {
        char digit = s.top();
        s.pop();
        cout << digit ;
    }

    return 0;
}
```

Q2. C++ Template (5 marks)

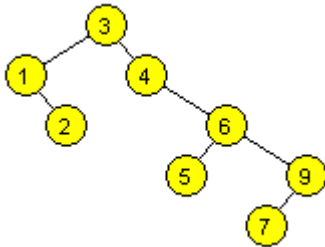
Given the following insertion sort function of an integer array:

```
void insertionSort(int arr[], int N) {  
    int i,j,tmp;  
    for (i=1; i<N; i++) {  
        tmp = arr[i];  
        for (j=i; j>0 && tmp < arr[j-1]; j--)  
            arr[j] = arr[j-1];  
        arr[j] = tmp;  
    }  
}
```

(a) (4 marks) Use C++ template to implement an insertion sort of an array of any type:

```
template <class T>    // 1 mark  
void insertionSort(T arr[], int N) {    // 2 marks  
    int i,j ;  
    T tmp ;    // 1 mark  
  
    // 1 mark of the correct for loop  
    // Copy and paste from above  
    for (i=1; i<N; i++) {  
        tmp = arr[i];  
        for ( j=i; j>0 && tmp < arr[j-1]; j--)  
            arr[j] = arr[j-1];  
        arr[j] = tmp;  
    }  
}
```

Q3. Tree transversal and node deletion in BST (10 marks)



(a) (4 marks) We define the tree transversal of BST as follows:

Transverse the right sub-tree

Transverse the left-sub-tree

Display the node value

What is the transversal result of the above BST?

_____ 7 9 5 6 4 2 1 3 (0.5 marks each) _____

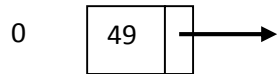
(b) (6 marks) Draw 2 different BSTs if a node 6 is deleted:

Method 1: Using the predecessor	Method 2: Using the successor
<p><u>Solution (3 marks):</u></p> <pre>graph TD; 3((3)) --> 1((1)); 3 --> 4((4)); 1 --> 2((2)); 4 --> 5((5)); 4 --> 9((9)); 5 --> 7((7));</pre>	<p><u>Solution (3 marks):</u></p> <pre>graph TD; 3((3)) --> 1((1)); 3 --> 4((4)); 1 --> 2((2)); 4 --> 7((7)); 4 --> 9((9)); 7 --> 5((5));</pre>

Q4. Hashing (10 marks)

Given input {49, 63, 64, 65, 66, 67, 68, 72, 75, 79, 82} and a hash function $h(x) = x \bmod 7$. Show the final result of the hash table using separate chaining. The first number 49 is given as an example:

Hash table Index



1

2

3

4

5

6

Solution: **Order is not important. For example,**

index 2 may have items 79 72 65 (instead of 65 72 79)

Index of the hash table							
0	49	63					
1	64						
2	65	72	79				
3	66						
4	67						
5	68	75	82				
6							

(10 marks , 1 mark each)

Q5. Binary Search Tree (10 marks)

In C++, a node of binary tree can be defined as follows:

```
struct Node {
    int value;
    Node *left;
    Node *right;
};
```

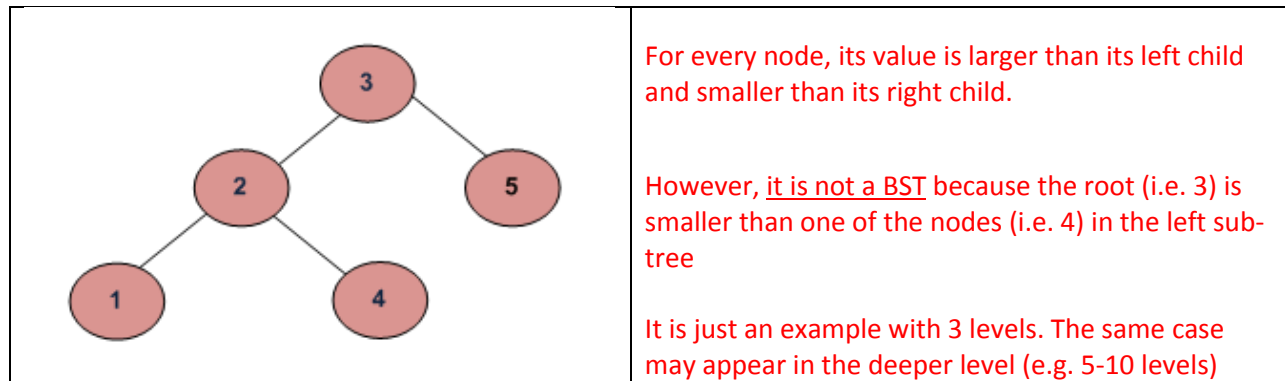
For simplicity, we assume the values in the binary tree are unique. The node values will be ranged from 1...10000 (inclusive)

Given a binary tree, check whether the tree is a binary search tree (isBST) or not.

Wrong idea

In each node, simply check the left child and right child, without considering all the remaining levels

Please note that you can't simply transverse every node and check whether its value is larger than its left child and smaller than its right child. A counter-example is given below:



For wrong idea, you may be able to get 0-3 marks as partial credits

Correct ideas:

1. In-order transversal and check the array is sorted or not
 - a. Use in-order transversal to visit every node, push_back all values to a vector object. After that, check whether the array is sorted or not (10 marks)
2. With the help of recursive functions
 - a. Implement the findMin and findMax recursive functions (6 marks)
 - b. The tree is not a BST if there exists: (4 marks)
 - i. the max node on the left sub tree (Note: not only a left child) is larger than the current node value OR
 - ii. the min node on the right sub tree is smaller than the current node value OR
 - iii. The left sub tree is not a BST OR The right sub tree is not a BST

For correct ideas, some partial credits will be given if the implementation is not 100% correct

Syntax error: -0.5 marks for each type (e.g. missing ";", using \geq instead of \geq)

Sample solution (for the idea 1 only)

```
// Write some helper functions here if needed

// Sample solution: Method 1 only
// It is impossible to list out all possible solutions
#include <iostream>
#include <vector>
void inorderFillArray(Node *btree, vector<int>& arr) {
    if ( btree != NULL ) {
        inorderFillArray(btree->left, arr);
        arr.push_back( btree->value );
        inorderFillArray(btree->right, arr);
    }
}
```

```
bool isBST(Node *btree) {

    // Write your isBST function here

    // Sample solution: Method 1 only
    // It is impossible to list out all possible solutions

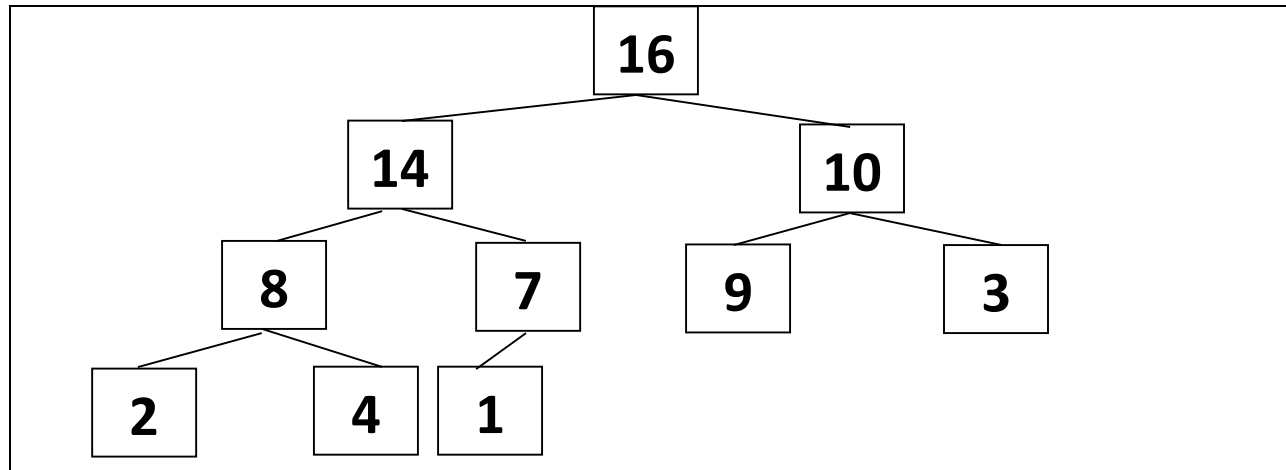
    vector<int> arr;
    inorderFillArray(btree, arr);

    // check the violation of sorted order
    for (int i=0; i<arr.size()-1; i++)
        if ( arr[i] > arr[i+1] )
            return false;
    return true; // sorted order, return true

}
```

Q6 Heapsort (15 marks)

A max-heap of 10 items is given as follows:



It is implemented on a pre-allocated array. We want to perform “delete Max” operation 10 times, such that finally the elements are sorted in increasing order. Fill in the arrays after each “deleteMax” operation. The array indices are given at the first line. The second line is the content of the max-heap.

<u>0</u>	<u>1</u>	<u>2</u>	<u>3</u>	<u>4</u>	<u>5</u>	<u>6</u>	<u>7</u>	<u>8</u>	<u>9</u>	<u>10</u>
	16	14	10	8	7	9	3	2	4	1
	14	8	10	4	7	9	3	2	1	16
	10	8	9	4	7	1	3	2	14	16
	9	8	3	4	7	1	2	10	14	16
	8	7	3	4	2	1	9	10	14	16
	7	4	3	1	2	8	9	10	14	16
	4	2	3	1	7	8	9	10	14	16
	3	2	1	4	7	8	9	10	14	16
	2	1	3	4	7	8	9	10	14	16
	1	2	3	4	7	8	9	10	14	16
	1	2	3	4	7	8	9	10	14	16

Q7. Generic Programming (20 marks)

You will implement a simple **MyMatrix** class using C++ template. **MyMatrix** uses the STL vector class to store the elements of a matrix. All matrix operations can be run in $m \times n$ dimension where m and n are positive integers. **MyMatrix.h** is given below:

```
// MyMatrix.h - The header file of class MyMatrix
#include <cmath>
#include <iostream>
#include <fstream>
#include <vector>
using namespace std;

template <class T>
class MyMatrix {
public:
    MyMatrix(int, int);                // part (a)
    const vector<T> & operator[] (int) const;    // part (b)
    MyMatrix<T> operator+ (MyMatrix<T> &);    // part (c)
    bool operator== (MyMatrix<T> &);          // part (d)
    int numCols() const;                    // part (e)

private:
    // Store the matrix elements
    vector< vector<T> > array;
};
```

(a) (4 marks) Implement a constructor that creates an empty matrix with m rows and n columns

```
template <class T>
// Add your code here ...

MyMatrix<T>::MyMatrix( int rows, int cols )
: array( rows ) // 1 initializer list (necessary)
{
    for( int i = 0; i < rows; i++ )
        array[i].resize( cols );
}

// 1 mark for the correct function header
// 1 mark for correctly using the for loop
// 1 mark for resizing the vector variable
```

Page 10 of 14

(b) (2 mark) Implement the subscript operator [] to extract the row of the matrix

```
template <class T>
// Add your code here ...

const vector<T> &MyMatrix<T>::operator[]( int row ) const{
    return array[row];
}

// 1 mark for the correct function header
// 1 mark for returning the correct value
```

(c) (6 marks) Implement the addition operator + to add another matrix (*Assume two matrices are in the same dimension*)

```
template <class T>
// Add your code here ...

MyMatrix<T> MyMatrix<T>::operator+ (MyMatrix<T> &m)
{
    MyMatrix<T> n = *this;
    MyMatrix<T> result(n.array.size(), n.numCols());

    for(int row=0; row<n.array.size(); row++)
        for(int col=0; col<n.numCols(); col++)
            result[row][col] = n[row][col] + m[row][col];

    return result;
}

// 1 mark for the correct function header
// 2 marks for correctly using the for loop
// 2 marks for calculating the correct value
// 1 mark for returning the correct value
```

Page 11 of 14

(d) (4 marks) Implement the comparison operator == to compare two matrices.

Hints: return true if two matrices are equal,. Otherwise return false

```
template <class T>
// Add your code here ...
bool MyMatrix<T>::operator==(MyMatrix<T> &m) {
    MyMatrix<T> n = *this;

    if(m.array.size() != n.array.size() || m.numCols() != n.numCols())
        return false;

    for(int i=0; i<n.array.size(); i++)
        for(int j=0; j<n.numCols(); j++)
            if(m[i][j] != n[i][j])
                return false;

    return true;
}
// 1 mark for the correct function header
// 2 marks for correctly using the for loop to check each element
// 1 mark for returning the correct value
```

(e) (4 marks) Implement the function numCols() to return the number of column in the matrix.

Hints: You can use vector::size() to return the size of vector

```
template <class T>
// Add your code here ...

int MyMatrix<T>::numCols( ) const{
    return array.size()? array[0].size( ) : 0;
}

// 1 mark for the correct function header
// 3 marks for returning the correct value
```

Q8. Polymorphism and Inheritance (25 marks)

Given the following program:

```
#include <iostream>
using namespace std;
class A {
public:
    A() {cout << "A Ctor" << endl ;}
    virtual ~A() { cout << "A Dtor" << endl; } // (b): Remove "virtual"
    virtual void foo() = 0;
    void bar() {cout << "A bar" << endl ;} // (b): Add "virtual"
};

class B: public A {
public:
    B(): A() { cout << "B Ctor" << endl; }
    ~B() { cout << "B Dtor" << endl ; }
    void foo() { cout << "B foo" << endl; }
    void bar() { cout << "B bar" << endl ;}
};

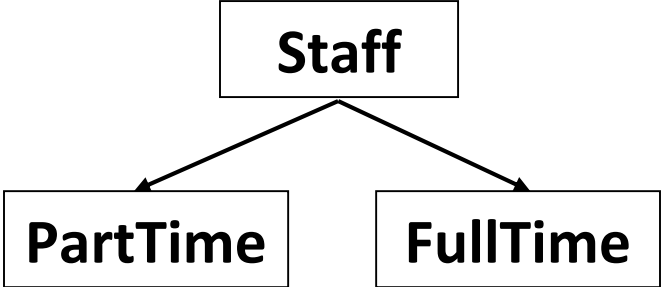
int main() { // (b): The main() function remains unchanged
    A *tmp = new B();
    tmp->foo();
    tmp->bar();
    delete tmp;
    return 0;
}
```

(a) (6 marks) What is the output of the program above?	(b) (5 marks) What is the output if <ul style="list-style-type: none">The keyword "virtual" is removed in the destructor of class A andThe keyword "virtual" is added in to the bar() function of class AThe main() function remains unchanged
A Ctor B Ctor B foo A bar B Dtor A Dtor (6 marks, 1 mark each)	A Ctor B Ctor B foo B bar A Dtor (5 marks, 1 mark each)

****Deduct 1 mark for each extra line**

Page 13 of 14

(c) (14 marks) Given the following class diagram and the formula to compute the salary of a part-time staff and a full-time staff respectively

Class Diagram	Formula to compute the salary
 <pre> classDiagram class Staff class PartTime class FullTime Staff < -- PartTime Staff < -- FullTime </pre>	<p>Part-time staff: Salary = hourlyWage * totalWorkHours</p> <p>Full-time staff: Salary = monthlyWage</p>

Please implement these classes with the following requirements:

1. The Staff class must be an abstract base class.
2. The member function "salary" must be implemented using polymorphism with "double" as the return type.

The main() function is given to illustrate the usage of these classes:

The main() function	Expected output
<pre> int main() { // The Base class "Staff" pointer Staff *p ; // p is now pointing to a PartTime object // works for 10 hours and \$200 hourly wage p = new PartTime(10, 200); cout << p->salary() << endl ; delete p; // p is now pointing to a FullTime object // monthly wage is \$400 p = new FullTime(400); cout << p->salary() << endl ; delete p; return 0; } </pre>	<pre> 2000 400 </pre>

Please complete the classes in the spaces provided on the next page. The implementation can be inside the class definition and you are not required to separate them.

Page 14 of 14

```
// Write the implementation of Staff, PartTime and FullTime classes
// in the spaces provided
```

(Note: int can be used to replace double)

```
class Staff {
public:

    Staff() {};
    virtual double salary() const = 0;

}; // Correctness of Staff class (4 marks)

class PartTime: public Staff {
public:

    PartTime(int h, int t) {
        hourlyWage = h;
        totalWorkHours = t;
    }
    double salary() const {
        return hourlyWage*totalWorkHours;
    }

private:
    double hourlyWage;
    double totalWorkHours;

}; // Correctness of PartTime (5 marks)

class FullTime: public Staff {
public:

    FullTime(int m) {
        monthlyWage = m;
    }
    double salary() const {
        return monthlyWage;
    }

private:
    double monthlyWage;
}; // Correctness of FullTime (5 marks)
```

=== END OF PAPER ===