Paper I

COMP 2012H Final Exam - Fall 2018 - HKUST

Date: December 13, 2018 (Thursday)
Time Allowed: 3 hours, 8:30am–11:30am
Instructions:

1. This is a closed-book, closed-notes examination.
2. There are **7** questions on **44** pages (including this cover page, appendix and 4 blank pages at the end) printed on **2** sets of papers:
   - Paper I: Description of problem 1 - 4 **AND** space for ALL your answers.
   - Paper II: Description of problem 5 - 7.
3. Write your answers in the space provided in paper I.
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, nor any library functions not mentioned in the questions.

Student Name	
Student ID	
Email Address	
Seat Number	

For T.A. Use Only

Problem	Topic	Score
1	True or False Questions	/ 10
2	Function Objects and STL	/ 8
3	Order of Construction and Destruction	/ 10.5
4	AVL Tree	/ 8.5
5	Inheritance, Polymorphism and Dynamic Binding	/ 26
6	Binary Search Tree (BST)	/ 17
7	Hashing	/ 20
Total		/ 100

Problem 1 [10 points] True or False Questions

Indicate whether the following statements are *true* or *false* by circling **T** or **F**.

T **F** (a) Functions can be overloaded on the basis of const-ness of parameters only if the const parameter is a pointer or a reference.

T **F** (b) Initialization of non-static data members in class will proceed according to the order of the member initialization list.

T **F** (c) `this` pointer is an implicit parameter to ALL member functions.

T **F** (d) C++ DOES NOT prohibit abstract base class from providing a definition for the pure virtual function.

T **F** (e) The type of inheritance (public, protected or private) does NOT change the accessibility of members inherited from the base class in the member functions of the direct derived class.

T **F** (f) There is NO compilation error in the following program AND it outputs `A's func`.

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

class A {
  public:
    virtual void func(int a = 20, int b = 10) { cout << "A's func" << endl; }
};

class B : public A {
  public:
    void func(int a, int b = 10) { cout << "B's func" << endl; }
};

int main() {
  A* p = new B;
  p->func();
  return 0;
}
```

T **F** (g) `typeid` only gives correct type information of the specified expression if it refers to a class that declares or inherits at least one virtual function.

T **F** (h) There is NO compilation error in the following program.

```
template <typename T>
class A {
  public:
    T funcWithSyntaxError() {
      int a = 10;
      int* p = a; // This line has syntax error
      return a;
    }
};

int main() { A<int> obj; }
```

T **F** (i) A unique binary search tree can be constructed from a given preorder traversal sequence.

T **F** (j) An AVL tree is balanced, therefore the median of all elements in the tree is always at the root or one of its children.

Problem 2 [8 points] Function Objects and STL

(a) [3 points] Define a function object class `Line` (in the file "`func-obj-line.h`") that will work with the given test program "`test-line.cpp`" to determine the y value of a given x value using line equation, y = mx + c, where m is the slope, and c is the y-intercept.

For example, running the test program will give the following output:

```
For line: y = 1.0 * x + 1.0, x = 5.0, y = 6
For line: y = 1.0 * x + 2.0, x = 6.0, y = 8
```

```
#include <iostream>     /* File: test-line.cpp */
using namespace std;

#include "func-obj-line.h" // Define function objects class Line in this file

int main()
{
  Line line1;
  double y1 = line1(5.0); // y1 = 1.0 * 5.0 + 1.0 = 6.0
  cout << "For line: y = 1.0 * x + 1.0, x = 5.0, y = " << y1 << endl;

  Line line2(1.0, 2.0);
  double y2 = line2(6.0); // y2 = 1.0 * 6.0 + 2.0 = 8.0
  cout << "For line: y = 1.0 * x + 2.0, x = 6.0, y = " << y2 << endl;
  return 0;
}
```

The constructor of `Line` will initialize such function objects with the slope and y-intercept, so that when the function object is called with an x value, it will compute the y value with its "memorized" slope and y-intercept.

Note that the `Line` class should only have

- two private data members,
- one constructor, and
- one overloaded operator function.

Answer:

File: func-obj-line.h

```
// Complete the class definition of Line here
```

(b) [3 points] Given the following prototype of the STL `transform` algorithm.

```
template <typename InputIterator, typename OutputIterator, typename UnaryOperation>
OutputIterator transform(InputIterator first, InputIterator last,
                         OutputIterator result, UnaryOperation Op);
```

implement the algorithm so that it applies operation `Op` to each of the elements in the range `[first, last)` and store the value returned by each operation in the range that begins at `result`. It returns an iterator pointing to the element that follows the last element written in the `result` sequence.

Answer:

```
template <typename InputIterator, typename OutputIterator, typename UnaryOperation>
OutputIterator transform(InputIterator first, InputIterator last,
                         OutputIterator result, UnaryOperation Op) {
  /* ADD YOUR CODE HERE */









}
```

(c) [2 points] Complete the following program in the space provided after "`TODO:`" comment lines so that it will run and gives the output below:

```
For line: y = 5x - 9, x = 1.8, y = 0
For line: y = 5x - 9, x = 2.4, y = 3
For line: y = 5x - 9, x = 6.7, y = 24.5
For line: y = 5x - 9, x = -23.1, y = -124.5
```

```
#include <iostream>
#include <vector>
#include <algorithm>
#include "func-obj-line.h"
using namespace std;

int main()
{
  vector<double> x;
  x.push_back(1.8);
  x.push_back(2.4);
  x.push_back(6.7);
  x.push_back(-23.1);

  vector<double> y(x.size());

  /*
   *  TODO: Using transform from part (b) together with a function object of
   *        Line defined in part (a), find the y values corresponding to the
   *        x values in container x and store the results in container y.
   *        Assume the line equation we use for this question is y = 5x - 9.
   */

  // ------------------------------ CODE HERE --------------------------------




  // -------------------------------------------------------------------------

  for(int i=0; i<x.size(); ++i)
    cout << "For line: y = 5x - 9, x = " << x[i] << ", y = " << y[i] << endl;
  return 0;
}
```

Problem 3 [10.5 points] Order of Construction and Destruction

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

class A {
  public:
    A() { cout << "A" << endl; }
    A(int a) { cout << "Conv A" << endl; }
    A(const A& a) { cout << "Copy A" << endl; }
    virtual ~A() { cout << "~A" << endl; }
};

class B : public A {
  public:
    B() { cout << "B" << endl; }
    B(const B& b) : A(b) { cout << "Copy B" << endl; }
    ~B() { cout << "~B" << endl; }
};

class C {
    static A a;
    static B b;
  public:
    C() { cout << "C" << endl; }
    C(int c) { cout << "Conv C" << endl; }
    C(const C& c) { cout << "Copy C" << endl; }
    virtual ~C() { cout << "~C" << endl; }
};

A obj(20);
A C::a(obj);

class D : public C {
  private:
    B b;
    A** a = new A*[2] { new A(b), new B(b) };
  public:
    D() { cout << "D" << endl; }
    D(int d) { cout << "Conv D" << endl; }
    D(const D& d) { cout << "Copy D" << endl; }
    ~D() {
      cout << "~D" << endl;
      for(int i=1; i>=0; i--)
        delete a[i];
      delete [] a;
    }
};

void process(const A aObj, const C cObj) { cout << "Processed" << endl; }
```

```
int main() {
  cout << "--- Block 1 ---" << endl;
  C cObj(D(10));
  cout << "--- Block 2 ---" << endl;
  process(10, 20);
  cout << "--- Block 3 ---" << endl;
  D dObj;
  cout << "--- Block 4 ---" << endl;
}
```

Write down the output of the above program when it is run. Some lines of outputs are already given. Assume the compiler DOES NOT do any optimization.

Answer:

```
--- Block 1 ---
--- Block 2 ---
--- Block 3 ---
--- Block 4 ---
```

Problem 4 [8.5 points] AVL Tree

(a) [7 points] Insert the following sequences of keys: 88, 60, 45, 99, 90, 120, 16, and 20 to an initially empty AVL tree. Draw all the intermediate trees (including the tree after insertion and each rotation, if any) and the final tree in the space provided below. You must use the algorithms discussed in class for inserting and re-balancing.

Answer:

(b) [1.5 points] Delete the key 90 from the final AVL tree obtained from part (a). Show all the intermediate trees (including the tree after deletion and each rotation, if any) and the final tree in the space provided.

Note: If the node to be removed has 2 children, replace the node with an appropriate value from the node's left sub-tree.

Answer:

Problem 5 [26 points] Inheritance, Polymorphism and Dynamic Binding

Please refer to Paper II for the problem description.

Based on the given information, complete the implementation of '`Banana`' class, '`Pineapple`' class and '`FruitSalad`' class in their respective .cpp files, namely "`Banana.cpp`", '`Pineapple.cpp`', '`FruitSalad.cpp`' respectively, and implement the missing operator functions in "`test-fruit.cpp`".

(a) [5 points] Complete the '`Banana`' class by implementing all the following member functions.

(i)
```
Banana(Color color, double weight, double calories,
       Size size, bool hasCut, double amountFiber);
```

Answer:
```
// File: Banana.cpp
```

(ii)
```
void cut();
```

Answer:
```
// File: Banana.cpp
```

(iii)
```
void print() const;
```

Answer:
```
// File: Banana.cpp
```

(b) [5 points] Complete the 'Pineapple' class by implementing all the following member functions.

(i) `Pineapple(Color color, double weight, double calories, Size size, bool hasCut, double percentageJuice);`

Answer: `// File: Pineapple.cpp`

(ii) `void cut();`

Answer: `// File: Pineapple.cpp`

(iii) `void print() const;`

Answer: `// File: Pineapple.cpp`

(c) [14 points]Complete the '`FruitSalad`' class by implementing all the following member functions.

(i) `FruitSalad();`

Answer: `// File: FruitSalad.cpp`

(ii) `FruitSalad(const FruitSalad& fs);`

Answer: `// File: FruitSalad.cpp`

(iii) `~FruitSalad();`

Answer: `// File: FruitSalad.cpp`

(iv) `FruitSalad& operator=(const FruitSalad& fs);`

Answer: `// File: FruitSalad.cpp`

(v) `FruitSalad& operator+=(const Fruit& f);`

Answer: `// File: FruitSalad.cpp`

(d) [2 points] Complete the "`test-fruit.cpp`" by overloading all the following operator functions.

(i) `ostream& operator<<(ostream& os, const Fruit& f);`

Answer: `// File: test-fruit.cpp`

```
ostream& operator<<(ostream& os, const Fruit& f) {
  // ADD YOUR CODE HERE




}
```

(ii) `ostream& operator<<(ostream& os, const FruitSalad& fs);`

Answer: `// File: test-fruit.cpp`

```
ostream& operator<<(ostream& os, const FruitSalad& fs) {
  // ADD YOUR CODE HERE




}
```

Problem 6 [17 points] Binary Search Tree (BST)

Please refer to Paper II for the problem description.

(a) [3 points] Implement

```
const T& find_lca(const T& x, const T& y) const;
```

to find the Lowest Common Ancestor of `x` and `y`. The lowest common ancestor of `x` and `y` is the shared ancestor of `x` and `y` that is located farthest from root.
Note: Recursion must be used for this question.

Answer: `/* File: bst-more.tpp */`

(b) [5 points] Implement

```
const T& find_ceil(const T& x, const T& minVal) const;
```

to find the ceiling of x. The ceiling of x is defined as the value equal to next greater key in the BST, if exists. If such a value does not exist, return `minVal`. If `x` is in the BST, then ceil is equal to `x`.

Note: Recursion must be used for this question.

Answer: `/* File: bst-more.tpp */`

(c) [3 points] Implement

```
T find_sum_left_boundary_nodes() const;
```

to find the sum of all nodes on the left boundary in a BST.
Note: Recursion must be used for this question.

Answer: `/* File: bst-more.tpp */`

(d) [3 points] Implement

```
T find_sum_right_boundary_nodes() const;
```

to find the sum of all nodes on the right boundary in a BST.
Note: Recursion must be used for this question.

Answer: `/* File: bst-more.tpp */`

(e) [3 points] Implement

```
T find_sum_boundary_nodes() const;
```

to find the sum of all nodes on the boundary in a BST.
Note: Recursion must be used for this question.

Answer: `/* File: bst-more.tpp */`

Problem 7 [20 points] Hashing

Please refer to Paper II for the problem description.

(a) [3 points] Implement

```
Hash(int capacity, int(*hashFunction)(int,int), int(*offsetFunction)(int,int));
```

below.

Answer: `// File: Hash.tpp`

(b) [0.5 point] Implement

```
~Hash();
```

below.

Answer: `// File: Hash.tpp`

(c) [6 points] Implement

```
int search(const KeyType& key, int& next) const;
```

below.

Answer: `// File: Hash.tpp`

(d) [4 points] Implement

```
void insert(T* data);
```

below.

Answer: `// File: Hash.tpp`

(e) [2 points] Implement

```
void remove(const KeyType& key);
```

below.

Answer: `// File: Hash.tpp`

(f) [4.5 points] Implement

```
void rehashing();
```

below.

Answer: `// File: Hash.tpp`

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