(COMP2012)[2015](s)final~schenax^_32832.pdf downloaded by lklamad from http://petergao.net/ustpastpaper/down.php?course=COMP2012&id=8 at 2023-02-04 09:53:43. Academic use within HKUST only.

COMP 2012 Final Exam - Spring Semester 2015 - HKUST

Date: May 26, 2015 (Tuesday)
Time Allowed: 3 hours, 4:30 – 7:30 pm
Instructions:

1. This is a closed-book, closed-notes examination.
2. There are **7** questions on **19** pages (including this cover page).
3. Write your answers in the space provided.
4. All programming codes in your answers must be written in ANSI C++.
5. Use *only* the C++ language features and constructs covered in the course so far.
6. 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.

Student Name	
Student ID	
Email Address	
Lecture & Lab Section	

Problem	**Score**
1. Function Object	/ 8
2. STL Iterator	/ 4
3. Hashing Analysis	/ 11
4. Open Addressing	/ 18
5. Sorting Algorithms	/ 18
6. Binary Tree & Inheritance	/ 23
7. Tree Traversal & Function Pointers	/ 18
Total	/ 100

(COMP2012)[2015](s)final~schenax^_32832.pdf downloaded by lklamad from http://petergao.net/ustpastpaper/down.php?course=COMP2012&id=8 at 2023-02-04 09:53:43. Academic use within HKUST only.

Problem 1 [8 points] Function Objects

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

#include "fcn-obj-pass.h"          // <— You need to write this file for the exam

int main( )
{
    vector<int> grade;
    grade.push_back(45);
    grade.push_back(88);
    grade.push_back(60);
    grade.push_back(15);

    for_each(grade.begin( ), grade.end( ), Pass(50));
    return 0;
}
```

Define function objects of the class `Pass` (in the file "sq-fobj-pass.h") that will work with the above main program to determine and to print out whether a given grade will pass or fail an exam. For example, running the above program gives the following output:

```
fail
pass
pass
fail
```

The constructor of `Pass` will initialize such function objects with a passing mark, so that when the function object is called with a grade, it will compare the given grade with its "memorized" passing mark and prints out "pass" when the given grade is greater than or equal to the passing mark or prints out "fail" otherwise. You may assume that all marks and grades are integers.

The `Pass` class should only have

- one private data member,
- one constructor, and
- one overloaded operator function.

(COMP2012)[2015](s)final~schenax^_32832.pdf downloaded by lklamad from http://petergao.net/ustpastpaper/down.php?course=COMP2012&id=8 at 2023-02-04 09:53:43. Academic use within HKUST only.

Answer:

// Complete the class definition of Pass in the file "fcn-obj-pass.h"

(COMP2012)[2015](s)final~schenax^_32832.pdf downloaded by lklamad from http://petergao.net/ustpastpaper/down.php?course=COMP2012&id=8 at 2023-02-04 09:53:43. Academic use within HKUST only.

Problem 2 [4 points] STL and Iterator

Convert the following code using a `for` loop and an `iterator` so that the code and your solution have identical output and effect when compiled and run. In the code, `x` is an STL vector of integers.

```
std::copy(x.begin( ), x.end( ), std::ostream_iterator<int>(std::cout, "\t"));
```

Answer:

(COMP2012)[2015](s)final~schenax^_32832.pdf downloaded by lklamad from http://petergao.net/ustpastpaper/down.php?course=COMP2012&id=8 at 2023-02-04 09:53:43. Academic use within HKUST only.

Problem 3 [11 points] Analysis of Separate Chaining for Hashing

Let $\alpha = N/M$ be the load factor to measure how full a hash table is, where N is the number of keys that have been inserted in the table, and M is the size of the table. With separate chaining, it is possible to have $\alpha < 1$, $\alpha = 1$, and/or $\alpha > 1$.

Given a load factor α, we investigate the time costs in the best-, average- and worst-case of

- new key insertion and unsuccessful find (these are the same)
- successful find

You can make the uniform hashing assumption in this question, and express your answers in terms of N, M, and α.

(a) The best-case time complexity for new key insertion and successful find are the same. What is it?

Answer: _____

(b) The worst-case time complexity for new key insertion and successful find are the same. What is it?

Answer: _____

(c) For the average case, we investigate the expected number of items per linked list. Now, we have N items distributed over M linked list (some of which may be empty).

i. In unsuccessful find/new-key insert, one of the linked lists in the table must be exhaustively searched. Let U_α be the average length of a linked list, which is equal to the average-time cost for unsuccessful find. What is U_α?

Answer: _____

ii. In successful find, the linked list containing the target key will be found. What is the average time cost S_α for successful find? Give a brief derivation for your answer.

Answer:

iii. Finally, show that U_α and S_α are $O(1)$ when $\alpha < 1$.

Answer: _____

(COMP2012)[2015](s)final~schenax^_32832.pdf downloaded by lklamad from http://petergao.net/ustpastpaper/down.php?course=COMP2012&id=8 at 2023-02-04 09:53:43. Academic use within HKUST only.

Problem 4 [18 points] Open Addressing for Hashing

Fill in the following hash tables to show their contents when different open addressing methods are used to insert the following 6 integer keys (in the given order) into the tables of size 7: 76, 9, 40, 47, 12, 55. The hash value of a key of value k is

$$hash(k) = k \bmod 7\,.$$

For each method, if a key cannot be inserted to the table after 10 probes, you may assume that the method fails. You may then stop (ignoring all remaining inputs) and describe what happens (that is, give the reason(s)).

(a) Linear probing.

Table Index	Insert 76	Insert 9	Insert 40	Insert 47	Insert 12	Insert 55
0						
1						
2						
3						
4						
5						
6						

(COMP2012)[2015](s)final~schenax^_32832.pdf downloaded by lklamad from http://petergao.net/ustpastpaper/down.php?course=COMP2012&id=8 at 2023-02-04 09:53:43. Academic use within HKUST only.

(b) Quadratic probing.

Table Index	Insert 76	Insert 9	Insert 40	Insert 47	Insert 12	Insert 55
0						
1						
2						
3						
4						
5						
6						

(c) Double hashing. Use the following $hash_2$ function as the 2nd hash function:

$$hash_2(k) = 5 - (k \bmod 5) \ .$$

Table Index	Insert 76	Insert 9	Insert 40	Insert 47	Insert 12	Insert 55
0						
1						
2						
3						
4						
5						
6						

(COMP2012)[2015](s)final~schenax^_32832.pdf downloaded by lklamad from http://petergao.net/ustpastpaper/down.php?course=COMP2012&id=8 at 2023-02-04 09:53:43. Academic use within HKUST only.

Problem 5 [18 points] Sorting Algorithms

Suppose the following list of 8 integers: 6, 0, 4, 5, 9, 8, 5, 0 are stored in an integer array `data` (of size 8) in the given order. Show the array content after each iteration/merge when the numbers are sorted in non-descending order by insertion sort, mergesort, and quicksort respectively. Specifically, show what are printed by the following `print_array` function when the respective sorting function is run. You should use the implementation codes of the algorithms given in our lecture notes; parts of them are given below for your reference.

```
void print_array(const int* data, int N)
{
    for (int j = 0; j < N; ++j)
        cout << data[j] << ' ';
    cout << endl;
}
```

(a) Insertion sort:

```
void insertion_sort(int* data, int N)
{
    for (int p = 1; p < N; ++p)
    {
        // Details of standard insertion sorting algorithm are assumed here
        print_array(data, 8);                // <— Show the content after each pass
    }
}
```

Answer:

(COMP2012)[2015](s)final~schenax^_32832.pdf downloaded by lklamad from http://petergao.net/ustpastpaper/down.php?course=COMP2012&id=8 at 2023-02-04 09:53:43. Academic use within HKUST only.

(b) Mergesort:

```
void mergesort(int* data, int* temp, int begin, int end)
{
    if (begin < end)
    {
        int center = (begin + end)/2;
        mergesort(data, temp, begin, center);
        mergesort(data, temp, center+1, end);
        merge(data, temp, begin, center, end);
        print_array(data, 8);              // <— Show the content after each merge
    }
}

void mergesort(int* data, int begin, int end)                    // Driver function
{
    int* temp = new int [end - begin + 1];
    mergesort(data, temp, begin, end);
    delete temp;
}
```

Answer:

(COMP2012)[2015](s)final~schenax^_32832.pdf downloaded by lklamad from http://petergao.net/ustpastpaper/down.php?course=COMP2012&id=8 at 2023-02-04 09:53:43. Academic use within HKUST only.

(c) Quicksort: Note that insertion sort will be called when an sub-array has 3 or less items. In addition, for this part, do **not** print the progress when insertion sort is called.

```
const int& median3(int* data, int begin, int end)
{
    int center = (begin + end)/2;

    if (data[begin] > data[center])
        std::swap(data[begin], data[center]);
    if (data[begin] > data[end])
        std::swap(data[begin], data[end]);

    if (data[center] > data[end])
        std::swap(data[center], data[end]);

    std::swap(data[center], data[end - 1]);
    return data[end - 1];
}

int partition(int* data, int begin, int end)
{
    const int& pivot = median3(data, begin, end);
    int left = begin;
    int right = end - 1;

    while (true)
    {
        while (data[++left] < pivot) { }
        while (data[--right] > pivot) { }

        if (left < right)
            std::swap(data[left], data[right]);
        else
            break;
    }

    std::swap(data[left], data[end-1]);
    return left;
}
```

(COMP2012)[2015](s)final~schenax^_32832.pdf downloaded by lklamad from http://petergao.net/ustpastpaper/down.php?course=COMP2012&id=8 at 2023-02-04 09:53:43. Academic use within HKUST only.

```
void quicksort(int* data, int begin, int end)
{
    if (end - begin > 2)
    {
        int pivot_index = partition(data, begin, end);
        quicksort(data, begin, pivot_index - 1);
        quicksort(data, pivot_index + 1, end);
    }
    else              // <— Don't show the printouts from insertion_sort this time
        insertion_sort(data + begin, end - begin + 1);

    print_array(data, 8);              // <— Show the content after each recursion
}
```

Answer:

(COMP2012)[2015](s)final~schenax^_32832.pdf downloaded by lklamad from http://petergao.net/ustpastpaper/down.php?course=COMP2012&id=8 at 2023-02-04 09:53:43. Academic use within HKUST only.

Problem 6 [23 points] Binary Tree, Inheritance and Class Template

We studied (mathematical) **expression trees** in this course which are represented by *binary trees* where leaf nodes are operands and non-leaf nodes are operators. *Inheritance* is a natural choice to implement the nodes in such hierarchical structure: let's have an abstract base class called `BT_node`, and derive operands and operators from it. In addition, to support operands of different types such as `int`, `double`, etc, the class for operands had better be implemented using C++ class *template*. However, you may assume that only C++ built-in types will be used for these operands. On the other hand, the derived class for operators should not be a class template in your implementation in this question.

In the following program, the function `build_tree` returns an expression tree. In this question, a tree is always represented by a pointer to a `BT_node` which is the root of the tree.

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

enum Optype { Add = 0, Sub, Mul, Div } ;
/* TODO: Definition of abstract base class BT_node */
/* TODO: Definition of derived class Operator */
/* TODO: Definition of derived class template Operand */

BT_node* build_tree( )
{
    BT_node* a = new Operand<double>(1);
    BT_node* b = new Operand<double>(2);
    BT_node* add = new Operator(Add, a, b);

    BT_node* c = new Operand<double>(3);
    BT_node* d = new Operand<double>(4);
    BT_node* subtract = new Operator(Sub, c, d);

    BT_node* e = new Operand<double>(5);
    BT_node* multiply = new Operator(Mul, e, subtract);
    return new Operator(Div, add, multiply);
};

int main( )
{
    BT_node* expr_tree = build_tree( );
    expr_tree->print( );
}
```

(COMP2012)[2015](s)final~schenax^_32832.pdf downloaded by lklamad from http://petergao.net/ustpastpaper/down.php?course=COMP2012&id=8 at 2023-02-04 09:53:43. Academic use within HKUST only.

(a) Draw the expression tree returned by `build_tree( )` in the above program. Use the convention in the textbook or in our lecture notes.

Answer:

(b) Write down the complete definition of the abstract base class `BT_node` which must satisfy the following requirements:

- it has only two private data members: the left and right subtrees.
- it has only one constructor which takes 2 input arugments with 0 as their default arguments to initialize the left and right subtrees. You must use member initializers wherever possible.
- it has 3 *const* member functions. Implement them as inline functions whenever possible.
  - `get_left_tree( )` returns the left subtree; it has no input arguments.
  - `get_right_tree( )` returns the right subtree; it has no input arguments.
  - `print( )` returns nothing and takes the depth of the tree as the only input argument which has a default value of zero. This function must be a *pure virtual function*. It prints the tree by rotating it by -90 degrees as done in our lecture notes. E.g., the expression tree of `5 + 6` should be printed as

```
        6
+
        5
```

(COMP2012)[2015](s)final~schenax^_32832.pdf downloaded by lklamad from http://petergao.net/ustpastpaper/down.php?course=COMP2012&id=8 at 2023-02-04 09:53:43. Academic use within HKUST only.

Answer: // Definition of the class `BT_node`

(c) Write down the complete definition of the class `Operator` which is derived from the abstract base class `BT_node` with the following requirements:

- it has only one private member called `my_op` of type `Optype`.
- it has a const member function `get( )` which returns its only data member.
- it has only one constructor function which takes 3 arguments: an `Optype` value, a left subtree, and a right subtree. You must use member initializers to initialize its own data member and any inherited data members.
- it must implement the pure virtual function `print( )` declared by its base class `BT_node`. It prints out the character operator (`+, -, *, /`) represented by its data member `my_op` preceded by a number of tabs as indicated by the depth argument. (That is, print d tabs if the depth is d.) This function must be defined outside the class definition.

(COMP2012)[2015](s)final~schenax^_32832.pdf downloaded by lklamad from http://petergao.net/ustpastpaper/down.php?course=COMP2012&id=8 at 2023-02-04 09:53:43. Academic use within HKUST only.

Answer: // Definition of the class Operator

(COMP2012)[2015](s)final~schenax^_32832.pdf downloaded by lklamad from http://petergao.net/ustpastpaper/down.php?course=COMP2012&id=8 at 2023-02-04 09:53:43. Academic use within HKUST only.

(d) Write down the complete definition of the class `Operand` which is derived from the abstract base class `BT_node` with the following requirements:

- it is a class template that has only one private data member, the `operand`.
- it has a const member function: `get( )` which returns the operand.
- it has only one constructor, taking only one argument (since it must be a leaf node with NULL left and right trees): an operand. Member initializer(s) must be used.
- it must implement the pure virtual function `print( )` declared by its base class `BT_node`. It prints out the operand preceded by a number of tabs as indicated by the depth argument. It must be defined outside the class definition.

Answer: // Definition of the class Operand

(COMP2012)[2015](s)final~schenax^_32832.pdf downloaded by lklamad from http://petergao.net/ustpastpaper/down.php?course=COMP2012&id=8 at 2023-02-04 09:53:43. Academic use within HKUST only.

Problem 7 [18 points] Function Pointers, Class Templates and Tree Traversal

In this question, you assume that a correct implementation of the previous question is available. Thus, even if you are not sure of your answer to the previous question, you should still attempt this question as much as you can.

The expression tree created in the previous main program may be evaluated by constructing an object of a new class called `Expr_eval`, making use of class templates and function pointers. The new main function is shown below and it outputs -0.6 when it is run.

```
int main( )
{
    Expr_eval<double> expression(build_tree( ));
    cout << expression.evaluate( ) << endl;
    return 0;
}
```

In other words, an `Expr_eval` object named `expression` is instantiated with the `double` type, and it is constructed with the expression tree created by `build_tree( )` defined in the previous problem. The expression tree is then evaluated by its member function `evaluate( )`. You may assume throughout this question that only `int`, `float` and `double` will be used to instantiate `Expr_eval`.

(a) Convert the following functions which only accept and output integer(s) to generic functions that may work on any types using C++class template.

```
int add(int x, int y) { return x+y; }
int subtract(int x, int y) { return x-y; }
int multiply(int x, int y) { return x*y; }
int divide(int x, int y) { return x/y; } // No need to check divide-by-zero
```

Answer:

(COMP2012)[2015](s)final~schenax^_32832.pdf downloaded by lklamad from http://petergao.net/ustpastpaper/down.php?course=COMP2012&id=8 at 2023-02-04 09:53:43. Academic use within HKUST only.

(b) Complete the following definition of the `Expr_eval` class template by completing the 3 TODO's.

```
enum Optype { Add = 0, Sub, Mul, Div } ;

/* Assume the correct definitions of BT_node, Operand, and Operator here */
template <typename T>
class Expr_eval
{
  private:    // There are 1 private member function and 2 private data members

    // A private utility function that is called by the public
    // driver function evaluate( )
    T eval(BT_node *) const;

    // TODO #1: Declare an appropriate data member to store an expression tree




    // TODO #2: Declare a private data member f, which is an array of function
    // pointers supporting the 4 operators as defined by enum Optype




  public:
    Expr_eval(BT_node* tree) : expr_tree(tree)
    {
        // TODO #3: Complete the constructor definition







    }

    // calculate( ) returns the value of  x op y  where op is +, -, *, or /
    T calculate(Optype op, T x, T y) const { return f[op](x, y); }

    // Call the private eval( ) to compute the result of the stored expression tree
    T evaluate( ) const { return eval(expr_tree); }
};
```

(COMP2012)[2015](s)final~schenax^_32832.pdf downloaded by lklamad from http://petergao.net/ustpastpaper/down.php?course=COMP2012&id=8 at 2023-02-04 09:53:43. Academic use within HKUST only.

(c) Complete the definition of the private utility member function `eval` of `Expr_eval` below, which implements a recursive tree traversal algorithm for expression evaluation. Only member functions of `Expr_eval` and the 2 library functions: `typeid` and `dynamic_cast` are allowed in your answer. You may find the following hints on the use of the 2 library functions useful.

```
string s = typeid(*node).name( ); // typename of (*node); node points to BT_node
s.find("Operator") != string::npos // true if typename s contains "Operator"
dynamic_cast<Operator*>(node)      // cast base class pointer (BT_node*)
                                   // to derived class pointer (Operator*)
```

```
template <class T>
T Expr_eval<T>::eval(BT_node* node) const
{
    if (node)                          // May assume that node is alway not NULL
    {                                  // TODO #4: Complete the definition of eval( )
    }
    else
    {
        cerr << "can't call with a NULL expression tree" << endl;
        exit(1);
    }
}
```

(COMP2012)[2015](s)final~schenax^_32832.pdf downloaded by lklamad from http://petergao.net/ustpastpaper/down.php?course=COMP2012&id=8 at 2023-02-04 09:53:43. Academic use within HKUST only.

/* Rough work — Don't detach this page as there is a question on the other side */

(COMP2012)[2015](s)final~schenax^_32832.pdf downloaded by lklamad from http://petergao.net/ustpastpaper/down.php?course=COMP2012&id=8 at 2023-02-04 09:53:43. Academic use within HKUST only.

/* Rough work — You may detach this page */

(COMP2012)[2015](s)final~schenax^_32832.pdf downloaded by lklamad from http://petergao.net/ustpastpaper/down.php?course=COMP2012&id=8 at 2023-02-04 09:53:43. Academic use within HKUST only.

/* Rough work — You may detach this page */