COMP 2012 Midterm Exam - Fall 2016 - HKUST

Date: November 5, 2016 (Saturday)
Time Allowed: 2 hours, 2–4pm
Instructions:

1. This is a closed-book, closed-notes examination.
2. There are **7** questions on **22** pages (including this cover page and the two rough worksheets).
3. Write your answers in the space provided in black/blue ink.
4. Allocate your time wisely. Show all your work to earn full credits.
5. All programming codes in your answers must be written in the ANSI C++ version as taught in the class.
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	Model Answer
Student ID	—
Email Address	—
Lecture Section	—

For T.A. Use Only

Problem	Score
1	/ 10
2	/ 9
3	/ 10
4	/ 10
5	/ 6
6	/ 25
7	/ 30
Total	/ 100

Problem 1 [10 points] True or false

Indicate whether the following statements are *true* or *false* by circling **T** or **F**.
1 point for each correct answer.

T **F** (a) If there is a user-defined copy constructor for class X, the compiler will not generate a default constructor (i.e. X(){}) for it.

T **F** (b) Member initialization list can only be used to initialize const and reference data members.

T **F** (c) Member functions that are defined within the class body are inline functions.

T **F** (d) The class destructor can receive arbitrary number of arguments.

T **F** (e) All the operators, / =, %, and ::, can be overloaded.

T **F** (f) A static member function cannot have access to both static and non-static members of class.

T **F** (g) A struct is the same as a class except there are no member functions.

T **F** (h) There is no difference between declaring a friend class in the public or private areas of the class.

T **F** (i) If a class A is friend of B, then B become friend of A automatically in C++.

T **F** (j) The following is an invalid template declaration.

```
template <typename T>
T* func() {
  return new T;
}
```

Problem 2 [9 points] Const-ness

```
#include <iostream>

class PrintProxy {
    int noPrint;
  public:
    PrintProxy() { noPrint = 0; }
    void print(char const* const p) const { std::cout << p << std::endl; }   // Line #7
    const int getConstNoPrint() const { return noPrint; }
    const int& getConstRefNoPrint() const { return noPrint; }
};

int main() {
  PrintProxy pp;
  char str[] = { 'C', 'O', 'M', 'P', '2', '0', '1', '2', '\0' };
  pp.print(str);
  int c_to_i = pp.getConstNoPrint();                    /* Error:   No   */
  const int c_to_ci = pp.getConstNoPrint();             /* Error:   No   */
  int cr_to_i = pp.getConstRefNoPrint();                /* Error:   No   */
  const int cr_to_ci = pp.getConstRefNoPrint();         /* Error:   No   */
  int& cr_to_ir = pp.getConstRefNoPrint();              /* Error:   Yes  */
  const int& cr_to_cri = pp.getConstRefNoPrint();       /* Error:   No   */
}
```

(a) There are 6 statements ending with the comments

```
/* Error:   Yes   /   No   */
```

decide whether the statement is syntactically INCORRECT – that is, it will produce compilation error(s). Circle "Yes" if it will give compilation error and "No" otherwise.

(b) In line #7, the const keyword appears three times. Briefly explain the meaning of each const. (from left to right)

First const: The first const means p is a pointer variable that points to a const char. It does not allow modification of the data through the pointer.

Second const: The second const means that p is a constant pointer, which the address stored cannot be changed.

Third const: The third const means print is a constant member function that guarantees it will not change any data members or call any non-const member functions.

Problem 3 [10 points] Member Initialization

(a) [3 points] Suppose we have a user-defined type `A` with default constructor. Is there any difference between the following two statements? Explain why.

```
A obj;
A obj();
```

They are different. The first statement: A obj; defines an object named obj of type A and the constructor invoked is the default one. The second statement: A obj() is the prototype of function named obj with no argument and the returned value is an object of type A.

Answer: _____

(b) [3 points] Is the default constructor for class A always A::A()? If not, explain why.

No, default constructor for class A does not have to be always A::A(). Constructor with default values could also be default constructor as well, for instance: A::A(int a = 10).

Answer: _____

(c) [2 points] Will a constructor of class A be called when we create an array of that type? If so, which constructor get invoked?

Yes, a constructor of class A will be called. The one gets called is default constructor.

Answer: _____

(d) [2 points] Suppose someone wants to initialize a const variable in primitive type within the class definition. Does it make sense? If not, in such case what else should the programmer do?

It doesn't make sense to initialize a const variable in primitive type within the class definition. In fact, it should be initialized using member initialization list.

Answer: _____

Problem 4 [10 points] Order of Construction & Destruction

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

class ExhaustFan {
  public:
    ExhaustFan() { cout << "E" << endl; }
    ~ExhaustFan() { cout << "~E" << endl; }
};

class Bathroom {
    string name;
    ExhaustFan exhaustFan;
  public:
    Bathroom(string n) { name = n; cout << "B" << endl; }
    ~Bathroom() { cout << "~B" << endl; }
};

class Kitchen {
    string name;
    ExhaustFan* exhaustFan;
  public:
    Kitchen(string n) : exhaustFan(new ExhaustFan) { name = n; cout << "K" << endl; }
    ~Kitchen() { delete exhaustFan; cout << "~K" << endl; }
};

class Apartment {
    Kitchen kitchen;
    Bathroom bathroom;
  public:
    Apartment() : bathroom("Warm bathroom"),
                  kitchen("Amazing kitchen") { cout << "A" << endl; }
    ~Apartment() { cout << "~A" << endl; }
};

int main() {
  Apartment* apartment = new Apartment;
  delete apartment;
}
```

What is the output of the above program?

E
K
E
B
A
∼A
∼B
∼E
∼E
∼K

Answer: _____

Problem 5 [6 points] Static Members

State whether the following program will compile. If yes, give the output. If no, give reason(s).

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

class Snoopy {
  public:
    Snoopy() { cout << "Snoopy's Constructor" << endl; }
    void sayHello() const { cout << "Snoopy says Hello!" << endl; }
};

class CSEFaculty {
  private:
    static Snoopy snoopy;
  public:
    static void playWithSnoopy() const {
      snoopy.sayHello();
    }
};

int main() {
  CSEFaculty::playWithSnoopy();
  return 0;
}
```

The program will not compile. First, static member function shall not be declared const. Second, the static object snoopy is not defined.

Answer: _____

Problem 6 [25 points] Class Template

This problem involves the implementation and application of a template class `Stack`. A stack is a data structure in which objects are added to and removed from the stack only at the top and it enforces Last-In-First-Out (LIFO) behaviour.

(a) Write a template class `Stack` that stores a list of objects using a dynamic array and provides the following data members and member functions:

- Data members
  - `maxSize` (a constant variable representing the maximum size of the dynamic array)
  - `data` (a pointer pointing to the dynamic array)
  - `topIndex` (representing the index of the top object, -1 if the stack is empty)
- Member functions
  - a constructor (allocating an array dynamically according to the specified size)
  - a destructor (deallocating the dynamic array)
  - `isEmpty()` (returning a Boolean value indicating if the stack is empty)
  - `isFull()` (returning a Boolean value indicating if the stack is full)
  - `push()` (inserting an object to the top of the stack, return true if the object is inserted successfully, otherwise return false)
  - `pop()` (removing the top object of stack and returning a pointer to the removed object, return NULL if the stack is empty)
  - `top()` (returning a pointer to the top object of the stack, without removing the object, return NULL if the stack is empty)
  - `size()` (returning the number of objects in the stack)

For simplicity, put all your function definition **INSIDE** the class template definition in a single file which will be called "Stack.h". Note that your solution should work with the given main function and produce the following output:

```
Push: Successful
Push: Successful
Top: -56.5
Pop: -56.5
Pop: 2.1
```

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

int main() {
  string str[] = { "Fail", "Successful" };
  Stack<double> stackD(5);                          // A stack of size 5

  // Push data
  bool s = stackD.push(2.1);
  cout << "Push:   " << str[s] << endl;
  stackD.push(-56.5);
  cout << "Push:   " << str[s] << endl;

  // Top and pop data
  double* ptr = stackD.top();
  if(ptr) cout << "Top:   " << *ptr << endl;
  else cout << "Stack is empty" << endl;
  ptr = stackD.pop();
  if(ptr) cout << "Pop:   " << *ptr << endl;
  else cout << "Stack is empty" << endl;
  ptr = stackD.pop();
  if(ptr) cout << "Pop:   " << *ptr << endl;
  else cout << "Stack is empty" << endl;
}
```

Answer: /* "Stack.h" */
// Implement the template class "Stack" here

```
template <typename T>
class Stack {
  private:
    int topIndex;
    T* data;
    const int maxSize;

  public:
    Stack(int ms) : maxSize(ms) {
      topIndex = -1;
      data = new T[ms];
    }

    ~Stack() {
      delete [] data;
    }

    bool isEmpty() const {
      return (topIndex == -1) ? true : false;
    }

    bool isFull() const {
      return (topIndex + 1 == maxSize) ? true : false;
    }

    bool push(const T& x) {
      bool retVal = false;
      if(!isFull()) {
        topIndex++;
        data[topIndex] = x;
        retVal = true;
      }
      return retVal;
    }
```

```
    T* top() const {
      T* retPtr = NULL;
      if(!isEmpty())
        retPtr = &data[topIndex];
      return retPtr;
    }

    T* pop() {
      T* retPtr = NULL;
      if(!isEmpty()) {
        retPtr = &data[topIndex];
        topIndex--;
      }
      return retPtr;
    }

    int size() const {
      return topIndex + 1;
    }
};
```

(b) In this part, you are required to use the template class `Stack` implemented in part (a) to evaluate a postfix expression. A postfix expression is an algebraic expression that requires its operators come after the two corresponding operands. The following shows a couple of postfix expressions:

- 2 3 + (Equivalent to 2 + 3)
- 1 2 3 * + 4 - (Equivalent to 1 + 2 * 3 - 4)

To evaluate a postfix expression using stack, we can use the following steps:

i. Create a stack to store operands

ii. Scan the given expression and do following for every scanned element
   - If the element is a number, push it into the stack
   - If the element is an operator, pop operands for the operator from stack, evaluate the operator and push the result back to the stack
   - When the expression is ended, the number in the stack is the final answer

Your task is to write a main function ("main-postfix.cpp") that prompts the user to input an postfix expression, evaluates the expression and output the result on screen. Include all the preprocessor directives that your program will use. The following shows the expected output of your solution:

```
Enter a postfix expression: 7 2 * 3 4 / - 5 +
Result: 18.25
```

You may assume that the user will always enter correct expression in the correct format, and you don't need to check for invalid inputs. Also the length of the postfix expression is less than 100.

Hints:

- getline() function can be used to get a line of string from the input console, e.g.,
  ```
  string str;
  getline(cin, str);
  ```
- The length of a string can be found using length() of string class, e.g.,
  ```
  string str = "Hello World";
  str.length(); // Length is 11
  ```
- operator[] can be used to get the character at certain position, e.g.,
  ```
  string str = "Hello World";
  char ch = str[1]; // ch = 'e'
  ```
- Conversion of numeric character to int can be done as follows:
  ```
  char ch = '9';
  int val = ch - '0'; // val = 9
  ```

Answer: /* "main-postfix.cpp" */
// Implement your main function here

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

int main() {
  string expression;
  Stack<double> stack(100);
  cout << "Enter a postfix expression:  ";
  getline(cin, expression);
  for(int i=0; i<expression.length(); i++) {
    char ch = expression[i];
    if(ch != '+' && ch != '-' && ch != '*' && ch != '/' && ch != ' ')
      stack.push(ch - '0');
    else if(ch != ' '){
      double* val1 = stack.pop();
      double* val2 = stack.pop();
      double result;
      switch(ch) {
        case '+': result = *val2 + *val1; break;
        case '-': result = *val2 - *val1; break;
        case '*': result = *val2 * *val1; break;
        case '/': result = *val2 / *val1; break;
      }
      stack.push(result);
    }
  }
  cout << "Result:  " << *(stack.pop()) << endl;
  return 0;
}
```

Problem 7 [30 points] Class and Operator Overloading

(a) The following shows a typical class definition for a Student. Complete the missing parts in the space provided under Part(a)(i)-(a)(iii) "`ADD YOUR CODE HERE`" by declaring (i) greater than operator `operator>`, (ii) less than operator `operator<` for the Student class, and making (iii) the insertion operator `operator<<` as a friend function of Student.

```
#ifndef STUDENT_H
#define STUDENT_H

#include <iostream>
using namespace std;

class Student {
    friend class School;
  private:
    string name;
    int yearOfStudy;
    string department;
    double CGA;
  public:
    void setStudent(string n, int y, string d, double c);

    // Declare greater than operator, operator>
    // Part (a)(i) - ADD YOUR CODE HERE
    bool operator>(const Student& s);

    // Declare less than operator, operator<
    // Part (a)(ii) - ADD YOUR CODE HERE
    bool operator<(const Student& s);

    // Make the non-member function, operator<<
    // a friend of Student class here:
    // Part (a)(iii) - ADD YOUR CODE HERE
    friend ostream& operator≪(ostream& os, const Student& s);
};

#endif
```

(b) Provide the implementation of the overloaded operators, (i) `operator<`, (ii) `operator>`, and `operator<<` for Student.

- Assume the result of comparisons between Students is based on the CGA value
- The output format of a Student object is as follows:

```
Name: John
Year: 4
Department: ACCT
CGA: 4.1
```

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

void Student::setStudent(string n, int y, string d, double c) {
  name = n; yearOfStudy = y; department = d; CGA = c;
}

// Greater than operator>
// Part (b)(i) Implement the member function here:
bool Student::operator>(const Student& s) {
  return (CGA > s.CGA);
}

// Less than operator<
// Part (b)(ii) Implement the member function here:
bool Student::operator<(const Student& s) {
  return (CGA < s.CGA);
}

// Part (b)(iii) Implement the non-member function operator<< here:
ostream& operator≪(ostream& os, const Student& s) {
  os ≪ "Name:   " ≪ s.name ≪ endl;
  os ≪ "Year:   " ≪ s.yearOfStudy ≪ endl;
  os ≪ "Department:   " ≪ s.department ≪ endl;
  os ≪ "CGA: " ≪ s.CGA ≪ endl;
  return os;
}
```

(c) Now suppose you need to help develop a simple student performance management system for a school using the Student class defined in part (a) and (b). To accomplish the task, a C++ class named `School` has been defined by the System Analyst of the school as follows:

```
#ifndef SCHOOL_H                                        /* School.h */
#define SCHOOL_H

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

class School {
  private:
      string name;
      Student* students;
      int maxNum;
      int curNum;
  public:
      School(string n, int max);
      School(const School& s);
      ~School();
      const School& operator=(const School& s);
      void addStudent(string n, int y, string d, double c);
      const Student* getHighestCGAStudent() const;
      Student** getLowerThanThreshold(double t) const;
};

#endif
```

Assume the member functions

```
School(string n, int max)
void addStudent(string n, int y, string d, double c)
```

have been implemented in `School.cpp` as shown on the next page.

```
#include <iostream>                                          /* School.cpp */
#include "School.h"
using namespace std;

School::School(string n, int max) {
  name = n; maxNum = max; curNum = 0;
  students = new Student[max];
}

void School::addStudent(string n, int y, string d, double c) {
  if(curNum < maxNum) {
    students[curNum].setStudent(n, y, d, c);
    curNum++;
  }
  else cout << "Reached maximum quota" << endl;
}

// *** 5 missing member functions: ***
// - School(const School& s)
// - ~School()
// - const School& operator=(const School& s)
// - const Student* getHighestCGAStudent() const;
// - Student** getLowerThanThreshold(double t) const;
```

Your task is to complete the remaining 5 member functions that satisfy the following requirements:

- Perform deep copy in the copy constructor and assignment operator function, i.e. `operator=`.
- Deallocate dynamic memory properly in the destructor
- The member function `getHighestCGAStudent()` should return a pointer to the student object which has the highest CGA among the list of student objects pointed by `students`
  (Hint: Make good use of `operator>` provided in the `Student` class)
- The member function `getLowerThanThreshold(double t)` should return a list of student objects which the CGA score is lower than the parameter value `t`. You could assume that the number of students with CGA lower than `t` is bounded by `maxNum`, and NULL should be put after the last object to indicate the end of the list if the number of returned objects is less than `maxNum`.
- The given testing program `main.cpp` will compile, run, and produce the output below.

```
#include <iostream>                                          /* main.cpp */
#include "School.h"
#include "Student.h"
using namespace std;

int main() {
  School school("Happy School", 10);
  school.addStudent("Peter", 2, "CSE", 3.2);
  school.addStudent("Mary", 3, "MATH", 3.5);
  school.addStudent("John", 4, "ACCT", 4.1);
  school.addStudent("William", 3, "IELM", 2.9);
  const Student* s = school.getHighestCGAStudent();
  if(s != NULL) {
    cout << "Student with the highest CGA" << endl;
    cout << "----------------------------" << endl;
    cout << *s << endl;
  }

  Student** list = school.getLowerThanThreshold(3.0);
  cout << "Student(s) with CGA < 3.0" << endl;
  cout << "-------------------------" << endl;
  for(int i=0; list[i] != NULL; i++)
    cout << *(list[i]) << endl;
}
```

Output of the testing program:

```
Student with the highest CGA
----------------------------
Name: John
Year: 4
Department: ACCT
CGA: 4.1

Student(s) with CGA < 3.0
-------------------------
Name: William
Year: 3
Department: IELM
CGA: 2.9
```

Answer: /* File "School.cpp" */
Implement all the missing member functions of the class `School` here:

```
School::School(const School& s) {
  students = NULL;
  *this = s;
}

School::~School() {
  if(students != NULL)
    delete [] students;
}

const School& School::operator=(const School& s) {
  if(this != &s) {
    if(students != NULL)
      delete [] students;
    maxNum = s.maxNum;
    curNum = s.curNum;
    students = new Student[s.maxNum];
    for(int i=0; i<s.curNum; i++)
      students[i] = s.students[i];
  }
  return *this;
}

const Student* School::getHighestCGAStudent() const {
  Student* student = NULL;
  if(curNum > 0) {
    student = &(students[0]);
    for(int i=1; i<curNum; i++) {
      if(students[i] > *student)
        student = &(students[i]);
    }
  }
  return student;
}
```

```
Student** School::getLowerThanThreshold(double t) const {
  Student** list = new Student*[maxNum];
  int n = 0;
  Student dummy;
  dummy.setStudent("", 0, "", t);
  for(int i=0; i<curNum; i++) {
    if(students[i] < dummy) {
      list[n] = &students[i];
      n++;
    }
  }
  // To indicate the end of the list
  if(n < maxNum - 1)
    list[n] = NULL;
  return list;
}
```