



## FINAL EXAMINATION

Course: **OBJECT ORIENTED PROGRAMMING**

Time: 100 minutes

Term: 3 – Academic year: 2018-2019

Student name:

Student ID:

*(Notes: Closed book exam)*

- Question 1.** a) Describe the difference between **private**, **public**, and **protected** in controlling the accessibility of object members in C++.
- b) What is destructor in C++? When and why do we need to explicitly define the destructor for a class?
- c) What is the difference between **overloading** and **overriding**?
- d) Reuse what is already written is one of the main purposes of OOP, why it is easier to reuse in C++ than in C?
- Question 2.** Assume all necessary libraries are included, read the C++ code below and answer the following questions:

```
01: class Chef {
02: public:
03:     virtual void prepare() = 0;
04:     void makeDish() {
05:         prepare();
06:         cout << "Made by Chef\n";
07:     }
08: };
09: class HeadChef: public Chef {
10: public:
11:     virtual void prepare() {
12:         cout << "Prepared by HeadChef\n";
13:     }
14: };
15: class SecondChef : public HeadChef {
16: public:
17:     void prepare() {
18:         HeadChef::prepare();
19:         cout << "Added by SecondChef\n";
20:     }
21: };
22: void makeFood(HeadChef c1, SecondChef c2) {
23:     c1.makeDish();
24:     c2.makeDish();
25: }
```



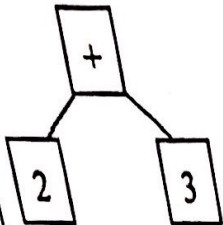
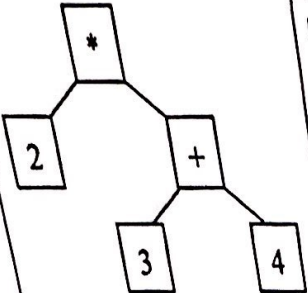
```
26: void main() {  
27:     SecondChef c;  
28:     makeFood(c, c);  
29:  
30:     Chef *c1;  
31:     c1 = new Chef;  
32:     c1->prepare();  
33:  
34:     c1 = new HeadChef;  
35:     c1->prepare();  
36:  
37:     c1 = new SecondChef;  
38:     c1->prepare();  
39:  
40:     SecondChef *c2 = new HeadChef;  
41:     c2->prepare();  
42: }
```

- a) Are there any lines in the main() function that cannot be compiled? Why can't they be compiled?
- b) Assume that all invalid lines of code are removed, what is printed to the screen after the line 28 is run?
- c) Assume that all invalid lines of code are removed, what is printed to the screen after the line 42 is run?
- d) At line 28, 'makeFood' takes two SecondChef objects as arguments, explain why we can do that.

**Question 3.** Consider a basic mathematical expression which is a series of real numbers and arithmetic operations (+, -, \*, /). A design option to represent this kind of expression is by using a tree.

There are two types of nodes in the expression tree:

- Number node: represents a number which has numerical value.
- Operation node: represents an operation which contains an operation symbol. Each symbol is either +, -, \*, or /. Each operation node contains a left node and a right node, which can either be number node or operation node.

Expression	Tree representation	Sample usage code
2 + 3		<pre>OpNode n('+'); n.addLeft( NumNode(2) ); n.addRight( NumNode(3) ); double x = n.evaluate(); // x = 5</pre>
2 * (3 + 4)		<pre>OpNode n1('+'); n1.addLeft( NumNode(3) ); n1.addRight( NumNode(4) );  OpNode n2('*'); n2.addLeft( NumNode(2) ); n2.addRight( n1 ); double x = n2.evaluate(); // x = 14</pre>

Applying encapsulation, inheritance and polymorphism in object oriented programming, you are asked to do the following:

- a) Draw a class diagram to show the tree representation above.  
The design should include necessary functions to construct an expression tree and evaluate the value of the expression.
- b) Write C++ code to implement the design.

\*\*\* GOOD LUCK \*\*\*