COMP 2150 TAKE-HOME EXAM – DUE APR 18, 8:59 AM

Rules for the take-home exam:

1. It is your responsibility to read all these rules completely.
2. You can use an IDE for this take-home exam. You can write code, test code and run code. You can use a debugger.
3. **You cannot submit any code you did not write yourself.**
4. You cannot discuss this question with **anyone** else. This includes people in the class and people not in the class. This includes people you don't know on the internet. Do not ask anyone any questions about any of the problems. "Talking" includes verbal communication with someone else or electronic communication of any form (text, images, sound, video, etc.) Do not talk to anyone.
5. You can refer to any class notes, videos or examples from class for this take-home exam.
6. You cannot search for solutions to this problem or any other problem online. You cannot submit any code you did not write yourself, even with a citation. Any code that is from any source other than yourself will be considered academic dishonesty. **Do not search for solutions.**
7. Any clarification requests should be sent to me by email. If I cannot reply (because it is not appropriate for me to answer the question in an exam), I will send back a very short email. Please don't take offense to how short it is. If I can reply, I will do so in a pinned thread on the forum – you will not get a response to your question by email.
8. Response time to a clarification will be under 24 hours. There will be no clarifications issued between 9 PM and 7 AM each night. Clarifications at other times will be responded to as soon as possible, but may be delayed due to other responsibilities I have. A lack of immediate clarification is not an excuse to seek unauthorized materials or engage in unauthorized discussions.
9. Any threads about the take-home exam (except the pinned thread) in the forum will be deleted. All clarification requests must come by email. Repeated posts on the forum about the take-home final will be considered a violation of academic integrity requirements for this take-home final, regardless of the contents of the post.
10. It is your responsibility to keep up with any clarifications in the pinned thread in the forum.
11. Comments made in your code may explain your code decisions. However, a comment with an incorrect assumption will not prevent deductions on that question.
12. The time for submission is 8:59 AM on Saturday, April 18. The time for submission is umlearn's clock, not your local system clock or any other clock. Give yourself time to submit the solution.
13. Late submissions will not be accepted. It is your responsibility to ensure that your submission is in on time.
14. Your submission should be a single zip file on umlearn. Other formats (including rar files or any other format) will not be accepted. The zip file must contain only the files required. No other files should be submitted (no readme file).
15. In the unlikely event that umlearn is down at or just before 8:59 AM on Saturday, April 18, an alternate mechanism to submit will be announced by email.
16. Academic dishonesty cases will be treated very seriously for this final exam. Academic dishonesty includes any violation of the conditions above or the Honesty Declaration for this course. This includes giving someone else **any** portion of your solution or **engaging in any discussions about the final exam.**
17. **The penalties for academic dishonesty are severe**: the typical penalty for cheating on an exam is an F in the course.
18. Failure to read these rules is not considered an excuse for academic dishonesty, or a valid reason for appeal.

Notes:

1. Apr 7, 7 AM. The example code for Question 1A has been updated (updates are in red).

Question 1

Consider the following Hashable class in C++:

```
class Hashable {
public:
        virtual ~Hashable();
        virtual int hashval() = 0;
        virtual bool operator==(const Hashable &) = 0;
};
```

You do not need to write any code for the Hashable class. You may assume that implemented subclasses of Hashable exist and implement the required pure virtual methods in an appropriate way. For example, you can assume that an implementation of IntHash exists that satisfies this interface:

```
#include "Hashable.h"

class IntHash : public Hashable {
private:
        int data;
public:
        IntHash(int);
        virtual ~IntHash();
        int hashval() override;
        bool operator== (const Hashable &) override;
        int getData();
};
```

Your code does not need this IntHash, and you do not need to code it. This example is only provided for your reference and to help you understand that Hashable interface (and to illustrate the result in parts A and B). You do not need to write any code for IntHash.

Consider also the following Node class:

```
class Hashable;

class Node {
private:
  Hashable* data;
  Node* next;
public:
  Node();
  Node (Hashable *i, Node* next);
  Node* getNext();
  Hashable* getData();
};
```

You may again assume that the class is properly implemented. You do not need to write any code for the Node class.

Consider also the following HashTable class:

```
class Node;
class Hashable;
class HashTableIt;

class HashTable {
        friend class HashTableIt;
private:
        Node** table;
        int size;
public:
        HashTable();
        void add(Hashable*);
        Hashable* get(Hashable*);
        bool contains(Hashable*);
        HashTableIt* iterator() const;
        bool operator== (const HashTable&);
 };
```

You may again assume that the HashTable class is properly implemented. In particular, you can assume that the constructor creates an array of a fixed size (initialized in the field called "size") and all of the elements of this array are initially set to nullptr. The add method will look like this:

```
void HashTable::add(Hashable* x) {
  int p = x->hashval();
  int spot = p % size;
  table[spot] = new Node(x,table[spot]);
}
```

In particular, note that there is no LinkedList class used to build the separate chains in this HashTable – nodes are used directly by the HashTable class in its implementation.

You only need to write one method for the HashTable class (in part B). The behaviour of add, get and contains are the same as in Assignment 4 – you can read that assignment for the intended actions.

[10 marks] A) Write an iterator for the HashTable. In particular (as shown in the HashTable interface) the iterator class must be called HashTableIt, and is a friend class. The constructor for HashTableIt should take one parameter (a **const** pointer to a Hashtable). It must use the Iterator design pattern (in particular, using the naming provided in the lecture material).

You are not required to provide any other code for this question. You should only submit two files – the HashTableIt.h file and the HashTableIt.cpp file.

For your reference, the following code would print out the four values 42, 1337, 9001 and 1042 in some order (the order is not important – any order that your code gives is okay).

```
#include "IntHash.h"
#include "HashTable.h"
#include "HashTableIt.h"
#include <iostream>
using namespace std;

int main () {
        HashTable* ht = new HashTable();
        ht->add(new IntHash(42));
        ht->add(new IntHash(1337));
        ht->add(new IntHash(9001));
        ht->add(new IntHash(1042));

        HashTableIt* it = ht->iterator();

        while (it->hasNext()) {
                IntHash* x = (dynamic_cast<IntHash*>(it->next()));
                if (x != nullptr) {
                        cout << x->getData() << endl;
                }
        }
        return 0;
}
```

[5 marks] B) Write the equality operator (==) for the HashTable class. The operator must be a method in the HashTable class. Submit only the code to implement the method. (You can assume this code is located in a cpp file. You should not provide the header file for HashtTable or any of the other code in the cpp file, only the operator method code).

Two HashTables are equal if they contain the same data. That is (as shown in the example below), every Hashable object in one HashTable should be found in the other, and vice-versa.

You must use the iterator from part A. If you are unable to finish part A, you can write the code for part B without a completed implementation to part A.

For your reference, the code shown below should print out 1 (for true):

```
#include "IntHash.h"
#include "HashTable.h"
#include "HashTableIt.h"
#include <iostream>

using namespace std;
int main () {

        HashTable ht;
        ht.add(new IntHash(0));
        ht.add(new IntHash(1000));
        ht.add(new IntHash(1));
        ht.add(new IntHash(2));

        HashTable ht2;
        ht2.add(new IntHash(1));
        ht2.add(new IntHash(2));
        ht2.add(new IntHash(1000));
        ht2.add(new IntHash(0));
        cout << (ht2==ht) << endl;
        return 0;
}
```

Question 2

[5 marks] Consider the Player interface (in Java) from Assignment 3

```
public interface Player {
  void lastMove(int lastCol);
  void gameOver(Status winner);
  void setInfo(int size, GameLogic gl);
}
```

In Assignment 3, you constructed two classes: one for a computer player and one for a human player. Write a Factory class for generating players of these two types, using the Software Factory pattern. Your factory must have only one method (the name of this method is up to you). That method should accept three parameters: a parameter that allows the client to select the type of player, a game logic pointer and the size of the board. The Factory method should return an object that is ready to have lastMove and gameOver called (that is, the objects returned from the method must be ready to play the game).

Note that the names of the classes that implement the computer and human player are up to you. Simply state them in your solution and assume that they are implemented. You do not need to submit code for those methods.

Submit one java file that contains the Factory.

Question 3

[5 marks] Create a hierarchy in JavaScript for hockey players. In hockey, we have two types of players and we will track information and statistics for both of them:

1. Goalies: goalies play in the net (or goal) and other players shoot at them. We will calculate each goalie's *save percentage*, which is the percentage of shots that are taken against them which are saved (that is, which are not goals). For example, if a goalie has 30 shots against them and one of them is a goal (and 29 are saved), their save percentage is .967. We must have the ability to track how many saves a goalie makes and how many goals are scored against them through appropriate method(s).
2. Skaters: all players other than goalies are skaters. These players shoot the puck to try and score. We will calculate the number of goals that each skater has scored. We must have the ability to track how many goals a skater has scored through appropriate method(s).

These two types of players are the only hockey players that should be able to be created. Additionally, each player (both skaters and goalies) will have a name and (possibly) a number associated with them. Each player (skater and goalie) needs to also be able to print a report for the player, which includes the name, number (if set), position (skater or goalie), number of goals (for skaters only) and save percentage (for goalies only).

For example, an appropriate report for a Goalie might look something like this (square brackets are just used to demonstrate where the name would go, do not include your own square brackets):

```
Player Name: [NAME]
Number: [NUM]
Position: goalie.
Save Percentage: 0.9875
```

Save percentages do not need to be rounded and arbitrary decimal places can be shown.

Errors should be reported by throwing an error. All printing should be done using console.log().

Each player needs to have a constructor. For both skaters and goalies, the name and (possibly) the number should be set when the object is created. If no number is given to a player, it should be set to -1 by default. The number -1 is never printed in a report: if the number is not set, it is not printed. The name is required, and it should be an error to attempt to create a player without a name.

You must use private JavaScript fields in this question. You must also use OO tools as appropriate, including encapsulation and refinement. You must also avoid code duplication. You do not need to provide any tools other than those listed above. You should submit your solution in a single js file.