INFOMGEP 2011	FINAL EXAM
Student name:	Student number:
Exam duration: 3 hours	Number of pages: 11

All the answers have to be written in the corresponding boxes.
No additional sheets can be handed in.
It is allowed to have:

- lecture notes with personal annotations
- course books

It is not allowed to have:

- other books
- non lecture related written material
- any printed material other than the lecture notes
- laptop, calculator and phone

Exercise 1 (1 point)

Draw a box-and-arrow figure of the memory state at the three following breakpoints.
Draw a box for each stack variable labeled with name and type, and containing its value when defined.
Draw a box for each object on the heap with its type, and value when defined.
Draw arrows to represent where pointers points to.

```
{ // enter scope
 double y = 14;
 double* x = new double[3];
 x[0] = 1; x[1] = 2; x[2] = y;
 double* z = x+1;
 double** w = new (double*)(z);
 // breakpoint 1
 // assuming Player::Player(double value):v(value){}
 // and Player::Player(double* value):ptrv(value){}
 delete w;
 Player p1 (y);
 Player p2 (x);
 Player * p3 = new Player(z);
 // breakpoint 2
 delete [] x;
} // exit scope
// breakpoint 3
```

At breakpoint 1

At breakpoint 2

At breakpoint 3

Exercise 2 (1 point)

For each question, check the answer(s) for which the statement is true. When multiple answers are possible, check all appropriate answers.

Question 2.1

The following program:

```
#include <iostream>
class A {};

template<class T>
class B {
    int i;
    public:
    B() :  i(0) {}
    void f(B<A>& o) { o.i = 1; std::cout << o.i;}
    T* g() { return new T();}
};

int main() {
    B<int> object1;
    B<A> object2;
    object1.f(object2);
    A* object3 = object2.g();
    return 0;
}
```

- ☐ prints "0"
- ☐ prints "1"
- ☐ does not compile, because
- ☐ generates a run-time error, because

Question 2.2

Does the following code compile?

```
A* a = NULL;
B* b = new B();
a = b;
```

- ☐ Yes
- ☐ No
- ☐ Yes, if `A` inherits from `B`
- ☐ Yes, if `B` inherits from `A`

Question 2.3

Assuming a class `MeshEntity`, the statement:
`MeshEntity meHuman1 = "human.mesh", meHuman2 = meHuman1;`

- ☐ returns false
- ☐ calls two different constructors of `MeshEntity`
- ☐ calls the assignment operator = of `MeshEntity`
- ☐ calls the copy constructor of `MeshEntity`

Question 2.4

Assuming the following program:

```
1  #include <iostream>
2  struct Player {
3
4  };

5  int main() {
6      Player p1;
7      std::cout << p1(2,3) << std::endl;
8      return 0;
9  }
```

What can you write at line `3` to make the program prints "5"?

- ☐ `int Player (int x, int y) { return x + y; }`
- ☐ `int operator() (int x, int y) { return x + y; }`
- ☐ `friend ostream& operator << (ostream& os, const Player& player) { os << player.x + player.y; }`
- ☐ `int p1 (int x, int y) { return x + y; }`

Exercise 3 (1.5 points)

In the Blackjack game of assignment 1, the deck received back and re-shuffled the whole set of cards after each round. Explain how you will modify your program to re-populate and re-shuffle the deck only when the number of unused cards is running low.

Illustrate your answer by giving at least the code:

- dealing with the cards used during a round (in `void Game::play()`)
- dealing with the threshold checking and associated actions (in `void Deck::deal(Hand& h)`)
- of any new data and function members you need to introduce

Reminder: `Deck`, `House` and `Player` inherit from `Hand`, and `Hand` has a protected vector of pointers to `Card` objects called `cards_`.

Exercise 4 (2.5 points)

In your game engine, you want to manage the `update` calls of your game entities (`AI` and `Physics` in this exercise). You want to design your engine so that only one `UpdateManager` instance can be created. The `UpdateManager` will manage (add and remove) the entities to notify. You have to make sure that an entity is stored uniquely (no multiple occurrences) in the `UpdateManager`. All entities to call will have an `update` function, specifically implemented according to the type of entity (you can refer it as `/* update code */` in this exercise). That function returns `false` if something wrong happened during the update, and takes a `double` as parameter representing the computer clock time (`clock()` call).

Give the declaration and implementation of all necessary classes (at least `UpdateManager`, `AI` and `Physics`). Give a main program that creates `AI` and `Physics` instances, uses the `UpdateManager` in a loop to update the entities and cleans up the program before exiting.

Exercise 5 (2.5 points)

In your game engine, you want to be able to create `GameEntity` objects from several sources (core program, plugin libraries, scripts, external resources etc.) through a regular factory:

```
class GameEntityFactory {
   public:
      GameEntity * createEntity (int entityID) const;
};
```

Creating ID dependent instances in `createEntity` requires to know the relationships between IDs and `GameEntity` objects. To avoid this, the factory will provide a registration mechanism (`register` and `unregister` functions). The factory will allow the (un)registration of `GameEntity` loaders. Registration is based on IDs provided by the loaders (you will assume that they are uniquely defined). `GameEntity` loaders will provide an interface to get IDs and to load `GameEntity` instances.

Give the declaration and implementation of the factory (does not have to be a singleton) and the loader classes for two different entities, as well as the program (un)registering loaders and creating instances of `GameEntity`.

Exercise 6 (1.5 points)

Question 6.1

If your `UpdateManager` class of exercise 4 is not thread-safe (regarding data members consistency and uniqueness of the manager), give a new implementation (only changes) that is.

Question 6.2

Rewrite the `createEntity` function of exercise 5 so that you throw an exception when the `entityID` is not recognized as a valid ID. Indicate also the changes you need to do to the main program.

Question 6.3

From exercise 5 framework, you want to write a `GameEntityFactory::printLoadersIDs()` function that creates a vector of loader IDs registered in the `GameEntityFactory`, then sorts it using a decreasing order and finally prints it to the screen (with ID values separated by `" - "` strings). Give the code of that function using the STL `sort` algorithm to sort the vector and that does not need any modifications to the loaders. Give two different versions of the call to the `sort` algorithm.