INFOMGEP 2011	Retake exam
Student name:	Student number:
Exam duration: 3 hours	Number of pages: 10

All the answers have to be written in the corresponding boxes.
It is allowed to have:
- lecture notes with personal annotations
- books

It is not allowed to have:
- 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 breakpoints in the following program.
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 point to.

```
class Player {
public:
    int level;
    double * life;
    Player(int lvl) :  level(lvl) {}
    Player(double * l) :  life(l) {}
    Player(const Player& p) {level = p.level;life = p.life;}
    Player& operator= (const Player& p) {level = 1;life = p.life;return *this;}
};

void main(int argc, char* argv) {
    int x = 2;
    double * y = new double(x);
    Player p1(x);
    Player p2(y);
    Player p3(p2);
    // breakpoint 1
    Player * p4 = &p3;
    delete p2.life;
    p3 = p2;
    // breakpoint 2
}
```

At breakpoint 1

stack	heap

At breakpoint 2

stack	heap

Exercise 2 (2 points)

Answer the questions and/or check the answer(s) for which the statement is true. When multiple answers are possible, check all appropriate answers.

Question 2.1

Assuming that the assignment operators of `Weapon` and `Inventory` might throw an exception, what is the highest exception safety level of the following `Player` assignment operator?

```
class Player {
public:
    Player& operator=(const Player& p) {
        try { w=p.w; i=p.i; }
        catch (...)  { }
        return *this;
    }
    Weapon w;
    Inventory i;
};
```

- ☐ Level 0: No guarantee
- ☐ Level 1: Destructibility
- ☐ Level 2: No resource leaks
- ☐ Level 3: Consistency
- ☐ Level 4: Full commit-or-rollback

Question 2.2

The following program:

```
class Game {
public:
    static Game* getInstance(int n){
        if (instance == NULL) {instance = new Game(); numPlayers = n;};
        return instance;
    }
private:
    Game(){};
    static Game* instance;
    int numPlayers;
};
Game* Game::instance = NULL;

void main() {
    Game* mygame = Game::getInstance(2);
}
```

- ☐ creates a `Game` instance with `numPlayers` equal to 0
- ☐ creates a `Game` instance with `numPlayers` equal to 2
- ☐ does not compile, because
- ☐ generates a run-time error, because

Question 2.3

In the following program, what can you write at line `16` to compile and execute the program without error? Indicate what the program prints for the adequate answer(s).

```
1  #include <iostream>
2  using namespace std;
3
4  class Enemy {
5  public:
6      virtual void attack() {cout << "Enemy attacks" << endl;}
7  };
8  class Ogre :  public Enemy {
9  public:
10     void attack() {cout << "Ogre attacks" << endl;}
11 };
12
13 void main() {
14     Enemy * ptrEnemy = new Enemy();
15     Ogre * ptrOgre;
16
17     ptrOgre->attack();
18 }
```

- ☐ `ptrOgre = ptrEnemy;` printing:
- ☐ `ptrOgre = (Ogre*) ptrEnemy;` printing:
- ☐ `ptrOgre = dynamic_cast<Ogre *>(ptrEnemy);` printing:
- ☐ `ptrOgre = static_cast<Ogre *>(ptrEnemy);` printing:

Question 2.4

The following program:

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

class Entity {
private:
    virtual void update() {cout << "Update entity, ";}
    void stop() {cout << "SE ";}
public:
    virtual void callUpdate() {cout << "Call update Entity, ";update();stop();}
};

class Player :  public Entity {
private:
    void update() {cout << "Update player, ";}
    void stop() {cout << "SP ";}
public:
    void callUpdate() {cout << "Call update Player, ";Entity::callUpdate();stop();}
};

void main() {
    Entity * p1 = new Player();
    p1->callUpdate();
    delete p1;
}
```

prints:

Exercise 3 (5 points)

Let's assume that initially your game engine allowed for only one player (class `Player`). You so designed it as a singleton:

```
class Player {
public:
    static Player* getInstance(){
        if (_instance == NULL) _instance = new Player();
        return _instance;
    }
private:
    Player(){};
    static Player* _instance;
};
```

Question 3.1 (1 point)

You now want to allow for a maximum of four players (number defined by `#define MAXPLAYERS 4`). Give the new implementation of the class `Player`, where the function `getInstance` is now replaced by `createNewInstance` and returns a new player if the current number of instances created is below the threshold and throws a STL exception of your choice otherwise. The current number of instances must also be decremented when necessary.

Question 3.2 (0.25 point)

If you did not used semaphore mechanisms, the function `createNewInstance` is not thread-safe. Explain (and illustrate with a scenario) what could go wrong.

Question 3.3 (0.25 point)

To store the player instances, you want to design a `PlayerManager` class. This class needs a container to store the players, named `_players`. It has a function to add a new player to the container. This function will call a function `PlayerManager::createPlayer` that itself calls `createNewInstance` of `Player`. Finally the class has a function to remove a player from the container and delete the instance. Among the STL containers, which is most adapted to that purpose, and why?

Question 3.4 (1 point)

Give the declaration of the class `PlayerManager` and the implementation of the function `createPlayer` (returning `NULL` if the creation failed).

Question 3.5 (0.5 point)

Both `PlayerManager` and `Player` have an interface to create `Player` instances. Explain why this can be an issue, and how you would solve it (without providing any code).

Question 3.6 (0.5 point)

You now want to manage update calls to player instances using the `PlayerManager`. You then decide to implement a function `updatePlayers` that iterates through the container and calls a function `update` on each element stored in the container.

```
void PlayerManager::updatePlayers() {
    // declaration of iterator it
    for (it = _players.begin(); it != _players.end(); ++it) {
        (*it)->update();
    }
}
```

But each player has its own frequency of update (in Hz), given by the function:
`double Player::getFrequency()`
The class `Player` also has the function `double Player::getLastUpdateTime()` returning the time at last update call. Give a new implementation of `updatePlayers` calling `update` at the right time for each player. You can use the statement `double t = GetTickCount();` to get the current computer time in **milliseconds**.

Question 3.7 (0.5 point)

In term of code exposition, better class designs exist. Propose a solution that does not require the class `Player` to expose its frequency and last update time (no code to provide).

Question 3.8 (1 point)

In your game engine, not only players need to be updated but many other types of object. Your goal is then to make the manager more generic (and now named `UpdateManager`).

Question 3.8.1 (0.25 point)

Give the declaration of a template based version of `UpdateManager`, where `Player` becomes any template type `T`.

Question 3.8.2 (0.25 point)

Give the declaration of an abstract interface based version of `UpdateManager`, where `Player` and the other types inherit from the base class `IUpdatable`.

Question 3.8.3 (0.5 point)

Indicate the main differences between both designs in their use, as well as their respective (dis)advantages regarding your goal.

Exercise 4 (2 points)

Assuming the following `Turret.h` declaration file:

```
#ifndef TURRET_H
#define TURRET_H

#include <utility> //for std::pair
#include <string>
#include "Player.h"
#include "BulletType.h"

class Turret {

public:
    Turret(Player* owner, int rateOfFire, BulletType btype);
    void setTarget(Player* target);
    void shootOneBullet();

private:
    std::pair<Player*,Player*> _ownerAndTarget;
    std::string _rateOfFire;
    BulletType _bulletType;

};

#endif
```

Question 4.1 (1 point)

Give a new version of the file `Turret.h` minimizing the number of `#include` directives.

Question 4.2 (1 point)

In your game logic, once created, a turret is an autonomous entity, shooting at the target at the given rate of fire. In order to model this behavior and to improve performance you want to dedicate one *shooting* thread per turret. Explain how you will implement it using Win32 threads. You can illustrate your explanation with pseudo-code for the `Turret` constructor and destructor, and the function executed by the thread.