

INFOMGEP 2012

RETAKE EXAM

Student name:

Student number:

Exam duration: 3 hours

Number of pages: 12

Number of points: 16

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 hand written material
- any printed material other than the lecture notes
- laptop, calculator and phone

EXERCISE 1 (2 points)

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.

```
struct PlayerParameters { int level; char * name; };

class Player {
public:
    PlayerParameters param;
};

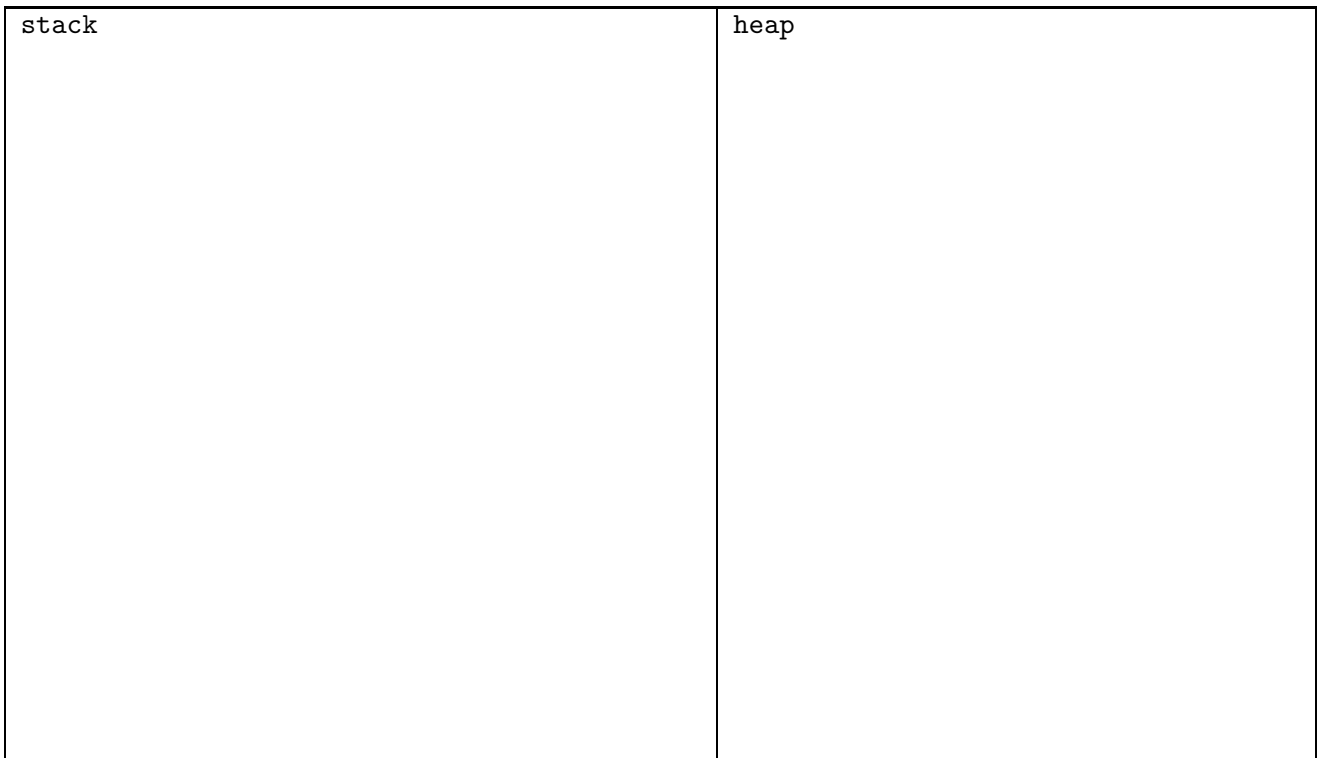
void destroyMe (Player * player) { delete player; };

void main() {
    Player * p1 = new Player();
    p1->param.level = 5;
    p1->param.name = new char [2];
    p1->param.name[0] = 'M'; p1->param.name[1] = 'e';
    Player p2;
    p2.param.level = 2;
    Player * p3 = p1;
    p3->param.level = p1->param.level / p2.param.level;
    // Breakpoint 1
    destroyMe(p3);
    // Breakpoint 2
};
```

At Breakpoint 1

stack	heap

At Breakpoint 2



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

Considering the following 2D array:

```
| int table [2] [2] = {{1,2},{3,4}};
```

Indicate the value of the variable x after execution of each of the following statements:

```
int x = * table[0];  
int x = * (table[0] + 1);  
int x = * (table[1] + 1);  
int x = * (table[2] - 1);  
int x = ** table;  
int x = ** (table + 1);
```

value of x:
value of x:
value of x:
value of x:
value of x:
value of x:

Question 2.2

Assuming the following class `Player`:

```
class Player {
public:
    Player () { level = 1; };
    Player (int lvl) : level(lvl) {};
    Player (const Player& player) {};
    void operator = (const Player& player) { level = player.level; };
    int level;
};
```

Indicate the value of the following variables after execution of the corresponding statements (indicate undefined if the value is undefined).

<code>Player p1;</code>	value of p1.level:
<code>Player p2(2);</code>	value of p2.level:
<code>Player p3(p1);</code>	value of p3.level:
<code>Player p4 = p2;</code>	value of p4.level:
<code>Player p5; p5 = p2;</code>	value of p5.level:

Question 2.3

Assuming the following classes:

```
1 | class Movable {
2 |     protected:
3 |         bool hasMoved;
4 | };
5 |
6 | class Updatable : public Movable {};
7 |
8 | class Drawable : public Movable {};
9 |
10 | class Player: public Updatable, public Drawable {
11 |     public:
12 |         bool hasMoved() const {
13 |
14 |         };
15 | };
```

Give the statement needed at line 13 in order to compile and properly run the function `hasMoved`.

Question 2.4

Assuming the following program:

```
1 | class Player {
2 | public:
3 |     int level;
4 |     bool operator < (const Player& player) { return level < player.level; }
5 |     bool operator () (const Player& player1, const Player& player2) {
6 |         return player1.level < player2.level;
7 |     }
8 |     bool isSmallerThan (const Player& player) { return level < player.level; }
9 | };
10 |
11 | bool comparePlayers(const Player& player1, const Player& player2) {
12 |     return player1.level < player2.level;
13 | };
14 |
15 | void main() {
16 |     vector<Player> players;
17 |     // ... construction of the vector of players (push_back) ...
18 |
19 | };
```

What can you write at line 18 to sort the vector of players by increasing level?

- `sort(players.begin(), players.end(), Player());`
- `sort(players.begin(), players.end(), &Player::isSmallerThan);`
- `sort(players.begin(), players.end(), std::mem_fun_ref(&Player::isSmallerThan));`
- `sort(players.begin(), players.end(), std::ptr_fun(&Player::isSmallerThan));`
- `sort(players.begin(), players.end(), &Player::comparePlayers);`
- `sort(players.begin(), players.end(), ptr_fun(comparePlayers));`
- `sort(players.begin(), players.end(), comparePlayers);`
- `sort(players.begin(), players.end());`
- `sort(players.begin(), players.end(), p1);`
- `sort(players.begin(), players.end(), Player);`

EXERCISE 3 (6 points)

In order to speed up the rendering of your game engine, you would like to integrate geometrical levels of detail (LoD). Geometrical LoD are multiple representations of the same game entity but that differ in complexity. For example, the avatar of the player can be a detailed virtual human or just a simple box. The rendering of a box will of course take much less time than a full virtual character, thus saving precious milliseconds at each game loop iteration.

Every game entity in the game engine should to be able to switch, at run-time, from one representation to another one. Your current declaration of the `GameEntity` class is as follows:

```

class GameEntity {
public:
    GameEntity(std::string name); // Constructor
    virtual ~GameEntity(); // Destructor

    virtual void update() = 0;
    // Update the entity, called by SceneNode::updateSceneNode
    virtual void draw() = 0;
    // Draw the entity, called by SceneNode::renderSceneNode

    std::string getName() const; // Get the name of the entity
    void setSceneNode(SceneNode * sceneNode);
    // Set the scene node to which the entity is attached
    SceneNode * getSceneNode() const;
    // Get the scene node to which the entity is attached
protected:
    std::string _name; // The name of the entity
    SceneNode * _sceneNode; // The scene node
};

```

Reminder: A scene graph is a tree structure of SceneNode objects on which GameEntity objects are attached. You can access the root node by calling the function SceneNode * getRootSceneNode() on the scene manager.

In your game engine, every game entity can have two representations (two LoD values), a low one and a high one. This information is private to the GameEntity class. You want to be able to set the LoD (low or high) of each entity individually.

Question 3.1 (1 point)

Indicate how you would modify the class GameEntity to manage the LoD value. Give the members you would add, remove or change.

Question 3.2 (1 point)

You want to manage the LoD internally in the `GameEntity` class at rendering time so that the LoD-dependent code will be executed only in derived classes by two different functions (one for each level). Give the declaration and implementation (C++ code) of the `GameEntity::draw` function. Indicate also the declaration of the functions it calls.

Declaration of `GameEntity::draw` and the functions it calls:

Implementation of `GameEntity::draw` :

Question 3.3 (1 point)

For the LoD to change at run-time, you need a criteria defining when to switch over. Let's suppose that you want to switch from high to low (resp. low to high) when the entity is further away from (resp. closer to) the camera than a threshold (defined by `LOD_DISTANCE`). You will assume that you have already programmed the function `vector<float> SceneNode::getAbsolutePosition()` returning the 3D absolute position of the scene node. You will also assume that you have programmed the global function `float getDistance(vector<float> position1, vector<float> position2)` returning the Euclidean distance between the two absolute positions.

Explain how and where you would implement the LoD switching so that every entity can automatically go back and forth between its high and low representations. Illustrate your answer with code or pseudo-code.

Reminder: the camera is a game entity unique in the virtual world and the only one attached to the root node of the scene graph.

Explanations:

Illustration with code or pseudo-code:

Question 3.4 (3 points)

Let's imagine that this optimization is not enough and you would also like to automatically switch to lower levels if the CPU performance decreases. To be efficient, you want to lower the entities that are the most consuming and the less critical. Therefore an entity has two additional parameters named `cost` (the higher, the more consuming to render) and `priority` (the higher, the more important to the gameplay). You want the LoD value to follow the rule illustrated in Figure 1, *i.e.* the LoD will be set to high only if the entity is not costly to render and important for the gameplay.

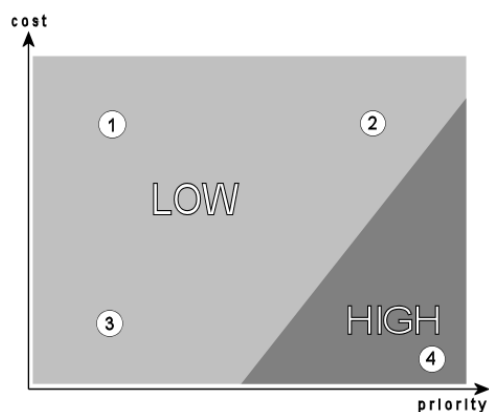


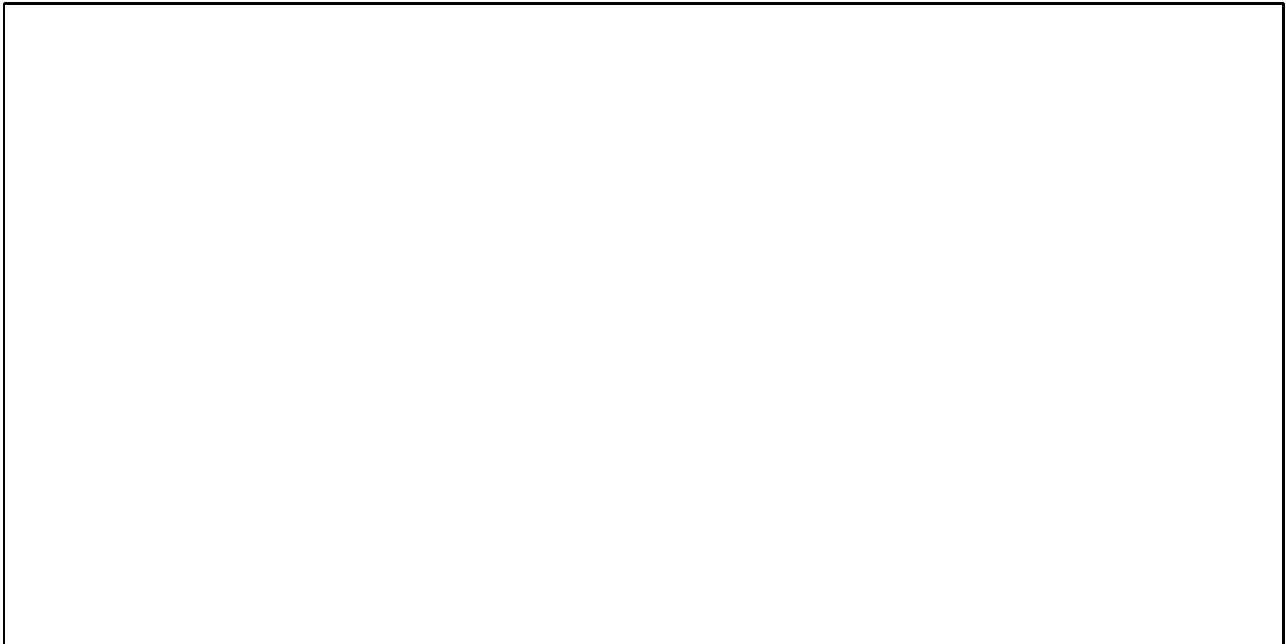
Figure 1: Value of LoD in the $(priority, cost)$ space.

Question 3.4.1 (0.5 points)

Give one typical example of a game entity belonging to each of the four areas noted in the figure (number 1 to 4). For example, the player will probably have by default a detailed avatar (very consum-

performance decreases (resp. increases). For the LoD value to change, the cost or the priority has to change. You will assume that the cost to render triangles and quads does not change over time (always the same time to render the same amount of polygons). So you want to dynamically change the priority of the entities. The slower the CPU will run, the lower the priority will be (and vice-versa). The priority of an entity will then linearly depend on the CPU performance, while clamped by user values (min and max) specific to this entity.

Give the code or pseudo-code to set an entity LoD level according to the CPU performance. You will assume that you have already programmed the function `int GameEngine::getFrequency()` returning the current FPS of the game.



EXERCISE 4 (6 points)

A useful feature for a game engine is the ability to manage particle systems. A particle system is concerned with manipulating amorphous objects like clouds of smoke, sparks, flames and so on. Such system is composed of a large number of relatively simple pieces of geometry facing the camera, the particles. A particle system is an entity in the world that creates and destroys the particles. The number of particles and their properties depend on the targeted effect.

Imagine that you want to manage three kinds of particle effects, namely fire (class `FireParticle`), smoke (class `SmokeParticle`) and waterfall (class `WaterfallParticle`), each having its own properties such as color, lifetime and animation.

Question 4.1 (4.5 points)

You want to create a class `ParticleSystem` to manage the creation and deletion of the particles. You want to simplify the management of the particles so that at the creation of the system all particles are created, and at the destruction of the system all particles are deleted. By default you would like to create systems of 500 particles.

Question 4.1.1 (1 point)

Your first idea is to design `ParticleSystem` as a template class, parameterized by the type of particle. Give the declaration and implementation of the template class `ParticleSystem`.

Question 4.1.2 (0.5 point)

Give the code to instantiate a template particle system of 500 fire particles, and to instantiate a template particle system of 200 smoke particles.

Question 4.1.3 (1 point)

Your second idea is to design `ParticleSystem` as a factory that stores and owns the particles. The type of particle will be determined by an ID. Give the declaration and implementation of the factory class `ParticleSystem`.

Question 4.1.4 (0.5 point)

Give the code to create a factory particle system of 500 fire particles, and to create a factory particle system of 200 smoke particles.

Question 4.1.5 (1.5 points)

List advantages and disadvantages of both designs regarding your goal.

Template:
advantages

disadvantages

Factory:
advantages

disadvantages

Question 4.2 (1.5 points)

In a particle system, every particle usually moves independently (no collision). To speed up the update of the position of the particles you decide to use multi-threading. As you do not want to create one thread for each particle (that will not speed up the update...), you decide that one thread will manage up to 100 particles.

Explain how you will modify the class `ParticleSystem` so that the update of the particles is done by such threads. Illustrate your answer by giving code or pseudo-code.

Explanations:

Illustration with code or pseudo-code on template based version: