1. Semistructured Data & XML (75 points)

The MegaDig store is specialized in the sale of music, films and games on numerical support. Note that we may have various formats for each kind of item, namely audio CD, video DVD, game for PC or MAC.

Let us assume the following XML DTD representing the catalogue of MegaDig store:

```xml
<!ELEMENT catalogue (audio, video, game)>
<!ELEMENT audio (CD*)>
<!ELEMENT video (DVD*)>
<!ELEMENT game (PC | MAC)*>
<!ELEMENT CD (title, price, (band | actor)?, duration, distribution)>
<!ELEMENT DVD (title, price, (actor)*, (director)?, duration, distribution)>
<!ELEMENT PC (title, price, distribution, version)>
<!ELEMENT MAC (title, price, distribution, version)>
<!ELEMENT title (#PCDATA)>
<!ELEMENT price (#PCDATA)>
<!ELEMENT duration (#PCDATA)>
<!ELEMENT distribution (#PCDATA)>
<!ELEMENT band (name, (contact)?)>
<!ELEMENT actor (name, (contact)?)>
<!ELEMENT director (name, (contact)?)>
<!ELEMENT version (#PCDATA)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT contact (#PCDATA)>
```

1.1. Is it possible to create a valid XML document containing CDs, DVDs and games, without any of the elements band, actor or director? (5 points)
   Yes

1.2. Is it possible to have in the catalogue an element CD after an element DVD, PC or MAC? (5 points)
   No

1.3. Is it possible to have in the catalogue an element MAC before a game for PC? (5 points)
   Yes

1.4. Is it the following XML document well-formed? Is it the following XML document valid with respect to the above DTD (justify your answers) (10 points)?
<catalogue>
  <audio>
    <CD>
      <title>Sgt. Pepper</title>
      <price>10</price>
      <band>
        <name>Beatles</name>
        <contact>G. Harrison</contact>
      </band>
      <duration>65</duration>
      <distribution>Polygram</distribution>
    </CD>
  </audio>
  <PC>
    <title>Lara Croft</title>
    <price>30</price>
    <distribution>Core and Eidos Interactive</distribution>
    <version>4</version>
  </PC>
  <MAC>
    <title>Afterlife</title>
    <price>25</price>
    <distribution>Lucasarts Demo</distribution>
    <version>2</version>
  </MAC>
</catalogue>

It is well-formed
It is not valid (the elements video and game are missing)

1.5. Simplify the above XML DTD by introducing appropriate XML attributes. These attributes make possible to eliminate some elements and simplify the content model of the remaining ones (give the new DTD with only non-terminal elements, i.e. omit the PCDATA elements) (20 points)?
<!ELEMENT catalogue (audio*, video*, game*)>
<!ELEMENT audio (title, price, (band | actor)?, duration, distribution)>
<!ATTRIBUTE audio kind (CD) CD>
<!ELEMENT video (title, price, actor*, director?, duration, distribution)>
<!ATTRIBUTE video kind (DVD) DVD>
<!ELEMENT game (title, price, distribution, version)>
<!ATTRIBUTE game kind (PC | MAC) PC>
<!ELEMENT band (name, (contact)?)>
<!ELEMENT actor (name, (contact)?)>
<!ELEMENT director (nom, (contact)?)>
<!ELEMENT (title | price | duration | distribution | ....) (#PCDATA)>
1.6. Give a valid instance of the XML DTD previously generated, which contains at least one example for each kind of element items in the catalogue (10 points).

```xml
<catalogue>
  <audio kind = CD>
    <title>Sgt. Pepper</title>
    <price>10</price>
    <band>
      <name>Beatles</name>
      <contact>G. Harrison</contact>
    </band>
    <duration>65</duration>
    <distribution>Polygram</distribution></audio>
  <video kind = DVD>
    <title>Blade Runner</title>
    <price>30</price>
    <actor>
      <name>Harrison Ford</name>
      <contact>Actcomp, Los Angeles</contact>
    </actor>
    <director>
      <name>Ridley Scott</name>
      <contact>Reacomp, Los Angeles</contact>
    </director>
    <duration>95</duration>
    <distribution>Warner</distribution></video>
  <game kind = PC>
    <title>Lara Croft</title>
    <price>285</price>
    <distribution>Core and Eidos Interactive</distribution>
    <version>4</version></game>
</catalogue>
```

1.7. Which of the following XML DTDs is well defined? (justify your answer) (20 points)

```
<!DOCTYPE family [
<!ELEMENT family (person*)>]
<!ELEMENT person (name, person, -- mother person, -- father person*,--children)>
<!ELEMENT name (#PCDATA)>]
```

```
<!DOCTYPE family [
<!ELEMENT family (person*)>]
<!ELEMENT person (name)>]
<!ATTLIST person id ID #REQUIRED
  mother IDREF #IMPLIED
  father IDREF #IMPLIED
  children IDREFS #IMPLIED>
<!ELEMENT name (#PCDATA)>]
```

Only DTD2 since the element `person` in DTD1 has an ambiguous content model.
2. Object-Oriented Databases (45 points)

The MegaDig store wants to use O2 in order to manage its catalogue. Read initially all the questions before starting to answer.

2.1. Create an O2 schema representing the DTD given in Section 1 (you can add new classes with different names from the XML elements). Note that the optional elements in the DTD can be represented by NULL values and an object schema also supports classes inheritance (20 points).

```o2
class Catalogue public type set(Item) end;
class Item public type
tuple(title: string,
   price: real,
   distribution: string)
end;
class Audio inherits Catalogue public type set(CD) end;
class CD inherits Item public type
tuple(artist: Artist,
   duration: integer)
end;
class Videos inherits Catalogue public type set(DVD) end;
class DVD inherits Item public type
tuple(actors: set(Actor),
   director: Director,
   duration: integer)
end;
class Games inherits Catalogue public type set(Game) end;
class Game inherits Item public type
tuple(version: integer)
end;
class PC inherits Game end;
class Mac inherits Game end;
class Artist public type
tuple(name: string,
   contact: string)
end;
class Band inherits Artist end;
class Actor inherits Artist end;
class Director inherits Artist end;

name CATALOGUE: Catalogue
```
2.2. Add a **Caddie** class which allows to represent customers’ shopping in the store (an object of this class makes it possible to manipulate a collection of **Items**) (10 points).

```plaintext
class Caddie public type set(Item) end;
```

2.3. For the class Caddie, implement a method **invoice**: real, which makes it possible to calculate the total amount to be paid. Note that the prices of the **Items** do not include taxes. All the products are taxed with 10%, except the games, which are taxed with 20% (Hint: It is necessary to also define methods in other classes of the schema) (15 points).

```plaintext
method body price: real in class Item {
   return self->price* 1.1;
}
method body price: real in class Game {
   return self->price * 1.2;
}
method body invoice: real in class Caddie {
   o2 article a;
   o2 real total = 0.0;
   for (a in (*self))
      total += a->price;
   return total;
}
```
3. Object Relational Databases (110 points)

Consider the following O2 schema (Text is one of the predefined multimedia types):

```java
class Film public
type tuple (title : string,
             review: Text);

class Theater public
type tuple (number: integer,
             film : Film);

class Cinema public
type tuple (name : string,
             address : string,
             theaters : set (Theater));

class Person public
type tuple (ssn: integer,
             name: string);

class Actor inherits Person public type
tuple (films: set (Film));

class Spectator inherits Person public type
tuple (likes: set (Film));

class Producer inherits Person public type
tuple (produces : set (Film));
```

name Persons : set (Person)
name Actors : set (Actor)
name Spectators : set (Spectator)
name Producers : set (Producer)
name Films : set (Film)
name Cinemas : set (Cinema)

3.1. Which is the condition so that any person in the Persons root is also either in Actors, either in Spectators, or in Producers (5 points)?

That each time I insert an actor (producer, spectator) in the Actors (Producers, Spectators) root, I also insert it as a person in Persons.
3.2. Given the above O2 schema express in OQL the following queries (40 points):

3.2.1. Which are the producers who are also actors?

```oql
select a
from a in Actors, p in Producers
where a = b
```

3.2.2. Which are the film titles, which we can currently see at a Cinema?

```oql
select s.film.title
from s in (flatten (select c.theaters
from c in Cinemas))
```

3.2.3. Which actors play in “Small Time Crooks”?

```oql
select a.name
from a in Actors, f in a.films
where f.title = “Small Time Crooks”
```

3.2.4. Find the title of films played by “Victoria Abril” and produced by “Almodovar”?

```oql
select f.title
from a in Actors, p in Producers, f in a.films
where a.name = “Victoria Abril” and p.name = “Almodovar”
and f in p.films
```

3.3. Create an object-relational (SQL3) schema according to the O2 schema given above. Note that tuple types correspond to ROW TYPES and inheritance is also defined for tables. You can ignore object identifiers for the purposes of this exercise (25 points).

```sql
CREATE ROW TYPE Film (title: string, review: Text);
CREATE ROW TYPE Theater (number: integer, film: Film);
CREATE ROW TYPE Cinema (name: string, address: string, theaters: set (Theater));
CREATE ROW TYPE Person (ssn: Integer, name: string);
CREATE TABLE Films OF Film
CREATE TABLE Cinemas OF Cinema
```
CREATE TABLE Persons OF Person

CREATE TABLE Actors UNDER Persons (films: set (Film))

CREATE TABLE Spectators UNDER Persons (likes: set (Film))

CREATE TABLE Producers UNDER Persons (produces: set (Film))

3.4. Given the previously generated object-relational schema express in SQL3 the following queries (40 points)

3.4.1. Which are the producers who are also actors?

```sql
select a.name
from Actors a, Producers p
where a.ssn = b.ssn
```

3.4.2. Which are the film titles, which we can currently see to the Cinema?

```sql
select f.title
from Cinemas c, Films f
where f.title in c.theaters→films.title
```

3.4.3. Which actors play in “Small Time Crooks”?

```sql
select a.name
from Actors a, Films f
where f in a.films and f.title = “Small Time Crooks”
```

3.4.4. Find the title of films played by “Victoria Abril” and produced by “Almodovar”?

```sql
select f.title
from Actors a, Producers p, Films f
where a.name = “Victoria Abril” and p.name = “Almodovar”
and f in a.films and f in p.films
```