Università degli Studi di Pisa

Laurea Specialistica in Informatica

Advanced Programming

Final Term Paper

Start Date: 4/07/2016
Submission deadline: 25/07/2016 (send a single PDF file to attardi@di.unipi.it)

Rules:

The paper must be produced personally by the student, signed implicitly via his mail address.

You are allowed to discuss with others the general lines of the problems, provided that each student eventually formulates his own solution. Each student is expected to understand and to be able to explain his solution.

You are allowed to consult documentation from any source, provided that references are mentioned.

It is not considered acceptable:

- **to consult or setup an online forum, to request help of consultants in producing the paper**
- to develop code or pseudo-code with others
- to use code written by others
- to let others use someone"s code
- to show or to examine the work of other students.

Violation of these rules will result in the cancellation of the exam and a report to the Presidente del Consiglio di Corso di Studio.

For the programming exercises you can choose a programming language among C++, C#, Java and JavaScript.

The paper must:

1. be in a single PDF file, formatted readably (**font size ≥ 10 pt** with suitable margins, single column), of **no more than 10 numbered pages**, including code: for each extra page one point will be subtracted from the score.
2. include the student name
3. provide the solution and the code for each exercise separately, referring to the code of other exercises if necessary.
4. cite references to literature or Web pages from where information was taken.

Introduction

APLIQ is a Domain Specific Language for dealing with collections of objects that provides a query facility similar to LINQ (Language Integrated Query). The APLIQ query language provides the following construct:

`item`	for defining a class of objects to be stored in a collection and the class of the corresponding `Collection`

For example:

```
item Book {
   [KEY] int ID;
   String title;
   String author;
```

```
      double price;
   }
   Collection<Book>      books = new Collection<Book>();
   Query<Book>           bookQuery = books.query();
```

The declaration `item` corresponds to a class definition whose fields are all `public`. The attribute `KEY` indicates that the field is to be considered a primary key.

`Collection<Book>`	class for a collection of items that are instances of class `Book`
`Query<Book>`	type of a query object that can be executed for returning a list of the items in a collection

Implementing the APLIQ involves a code generator that produces class `Book`, and providing classes `Collection` and `Query` implementing respectively the following interfaces:

```
   interface IBook {
      Book(int ID, String title, String author, double price);
      int ID;
      String title;
      String author;
      double price;
   }

   interface ICollection<T> {
      T get(int);
      void insert(T);
      IQuery<T> query();
   }

   interface IQuery<T> {
      List<T> execute();      // performs the query and returns the
                                                 // list of results
   }
```

The generated classes can be used in the following way:

```
   Collection<Books> books = new Collection<Book>();
   books.insert(new Book(1, "Iliad", "Homer", 9.99));
   books.insert(new Book(2, "Odissey", "Homer", 10.99));
   List<Book> allBooks = books.query().execute();
   for (Book book : allBooks) // Java syntax
     System.out.printl(book);
```

Exercise 1

Write a LL(1) grammar for the APLIQ `item` definitioms.

Exercise 2

Define suitable classes to represent `item` definitions expressed in APLIQ.

Exercise 3

Write a recursive descent parser for the APLIQ that produces a representation of the input using the classes of Exercise 2.

Exercise 4

Write a code generator that exploits polymorphism to produce classes and methods that implement the APLIQ.

Exercise 5

Extend the APLIQ providing the ability to specify a condition on queries with the following syntax:

`query.where(CONDITION)`	defines a query object that returns a list of elements of a container satisfying the query

The clause `CONDITION` provides conditions that items of the collection must fulfill to be listed among the results. A clause is made as a conjunction of elementary clauses, which consists of comparisons between fields, for instance:

```
query<Book> homerBooks = books.query().where(author == "Homer" and
                     price < 10.0);
```

Allow using variables in the clauses, like:

```
query<Book> homerBooks = books.query().where(author == "Homer" and price
                     < x);
```

Consider the following possible relations: ==, <, <=, >, >=, `!=`.

Provide a parser for these query expressions and code generator that exploits inheritance and polymorphism:

Show the code generated for the examples above.

Exercise 6

Add a clause for ordering the results, for example:

```
query<Book> homerBooks = books.query().where(author == "Homer" and
                     price < 10.0).sort(title);
```

Show the code generated for the example.

Esercise 7

Explain differences and similarities between arrays and objects in the JavaScript object model. What is the difference between a prototype in JavaScript and a class in other languages?