Linguagens e Ambientes de Programação (2019/2020) [Eng.Inf. - DI/FCT/UNL]

Enunciado do 1º Projeto Prático (OCaml)

Artur Miguel Dias

Datas

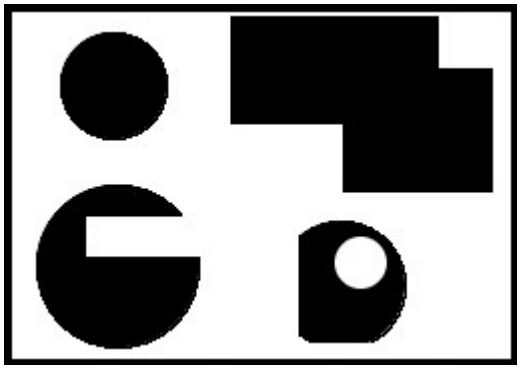
- 27/mar (20:00) - Divulgação da versão provisória do enunciado
- 30/mar (09:00) - Divulgação da versão final do enunciado
- 04/abr (20:00) - Data e hora limite de entrega do 1º projeto.
- 08/abr (20:00) - Data e hora limite de entrega com atraso. Um valor de penalização por cada dia de atraso.

Changelog

- 28/mar (12:00) - No ficheiro Shape.ml oferecido, havia alguns cabeçalhos de função com **rec** e outros sem **rec**. Esses **rec** estavam lá por distração e foram todos apagados. O **rec** é um aspeto da implementação (que nem sequer altera o tipo da função) e compete aos alunos decidir se precisam ou não de usar o **rec**.
- 27/mar (20:00) - Possíveis correções a este enunciado serão assinaladas aqui.

Shapes

Please analyze the following two-dimensional shape, drawn in black on a white background:



It is a geometric shape that groups four subshapes that you be able to distinguish well:

1. The first subshape, at the top left, is a basic shape, specifically a circle.
2. The second subshape, at the top right, results from the union of two horizontal rectangles.
3. The third subshape results from the subtraction between a circle and a horizontal rectangle.
4. The fourth subshape, the most complex, results from the intersection of a circle with a horizontal rectangle, followed by the subtraction of a small inner circle.

Module "Shape"

The aim of this project is to write a closed OCaml module named "Shape" containing a data representation for complex two-dimensional geometric shapes and the implementation of some functions over those shapes.

The module interface has already been fully written and you are not allowed to change it: Shape.mli. As you can see, the data representation is public and there is also a small number of public functions declared. All the other entities you might define in the module body will become automatically private. The representation is public to allow Mooshak to check the program deeply.

Use this file as a starting point to write your module body: Shape.ml.

Characterization of the shapes

We consider two broad categories of shapes:

- **Basic shapes** - Circle and (horizontal) Rectangle.
- **Binary composite shapes** - Union, Intersection and Subtraction.

Regarding the basic shapes, some details:

- Each **Rectangle** is characterized by its left top (a 2d-point) and its right base (another 2d-point).
- Each **Circle** is characterized by its center (a 2d-point) and its radius (a real number).
- The basic shapes are closed, that is, the border of a basic shape is part of the shape itself.

Regarding the composite shapes:

- Each composite shapes are characterized by two subshapes that we call "the left subshape" and "the right subshape".
- A composite shape can be partially or totally open because of the Subtraction variant. When we subtract a closed shape, part of the border of the remaining shape can be taken away. This happened twice in our opening example.

For the representation of Shapes, we use the following OCaml types:

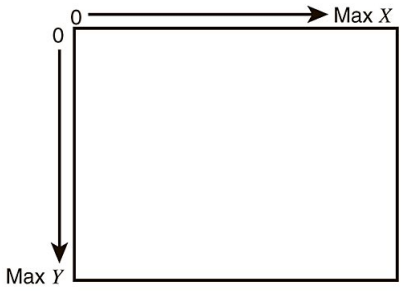
```
type point = float*float;;

type shape = Rect of point*point
           | Circle of point*float
           | Union of shape*shape
           | Intersection of shape*shape
           | Subtraction of shape*shape
;;
```

To exemplify, the constant `shape1` below is is a very simple composite shape:

```
let rect1 = Rect ((0.0, 0.0), (5.0, 2.0));;
let rect2 = Rect ((2.0, 2.0), (7.0, 7.0));;
let shape1 = Union (rect1, rect2);;
```

In our coordinate system, the x-axis goes from left to right and the y-axis goes from top to bottom:



The public functions of the module

There are ten public functions to implement. Please, write the functions using the inductive method and avoid any kind of imperative code. Also, take advantage of the predefined functions of the module List in case you use lists in some functions.

Restricting yourself to shapes made up only of Rectangles and Unions, makes these functions very approachable. On the other hand, the other three variants can complicate some functions and, in some cases, even require some research (specially in the case of the last function). Furthermore, some cases may be even impossible or open problems, and it is up to you to determine this. If you do not want to deal with a particular variant (say `Subtraction (l,r)`) in a particular function, you should call `failwith` to raise an exception.

→ Deixa a nosso critério!

1. **`hasRect : shape -> bool`**
   `let hasRect s = ...`

   Checks if a shape **s** contains any rectangle in its structure.

   Already solved, as an example:

```
let rec hasRect s =
    match s with
          Rect (p,q) -> true
        | Circle (a,f) -> false
        | Union (l,r) -> hasRect l || hasRect r
        | Intersection (l,r) -> hasRect l || hasRect r
        | Subtraction (l,r) -> hasRect l || hasRect r
;;
```

2. **`countBasic : shape -> int`**
   `let countBasic s = ...`

   Counts the number of basic shape components in a shape **s**. For example, in the shape of the opening example there are eight basic shapes.

3. **`belongs : point -> shape -> bool`**
   `let belongs p s = ...`

   Tests whether the point **p** belongs to the black part of the shape **s**.

4. **`density : point -> shape -> int`**
   `let density p s = ...`

   Determines the density of a shape **s** at a point **p**.

   The density at **p** is zero in a white area. The density at **p** is greater than zero in a black area.

   The density at **p** is the number of overlays of basic shapes at **p** that are not canceled out by intersections and differences. In the second subshape of our initial example, the density is two in the

overlapping area of the two rectangles; in the fourth subshape, the density is also two in the intersection area of the larger circle with the rectangle, except for the area of the small circle.

More detail:

- In a **basic shape**, the density is one if **p** belongs to the shape; zero outside
- In a **union**, the two partial densities are added up if **p** is in the intersection zone; outside that common area the density remains.
- In an **intersection**, the two partial densities are added up if **p** is in the intersection zone; but the density is zero if **p** is outside the intersection zone.
- In a **difference**, the density remains if **p** is in the part of the shape that remains; the density is zero if **p** is in the part that represents erasure.

The function **density** can use the function **belongs**, but the other way round is not allowed.

5

```
which : point -> shape -> shape list
let which p s = ...
```

Determines what are the exact basic subshapes of a shape **s** that contributed to the density at a point **p**. For example, if the density at a point **p** is 4, then the result should be a list of the 4 basic subshapes that caused the result 4. The order of the result is not important.

The function **which** can use the functions **belongs** and **density**, but the other way round is not allowed.

6

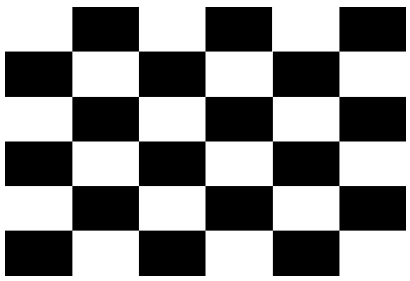
```
minBound : shape -> shape
let minBound s = ...
```

Calculate the the **minimum bounding rectangle** that contains all the basic subshapes in a shape **s**. Another common name for this rectangle is **bounding box**.

7

```
grid : int -> int -> float -> float -> shape
let grid m n a b = ...
```

Creates a shape that represents a two-dimensional grid of **m** * **n** rectangles, all with size **a** * **b** and touching each other. The top-left rectangle is white and touches the origin of the coordinate system. The grid below has been created using **grid 6 6 1.5 1.0**. Note that **m** and **a** pertain to the x-axis and **n** and **b** pertain to the y-axis.



8

```
countBasicRepetitions: shape -> int
let countBasicRepetitions s = ...
```

Given a shape **s**, counts the number of repetitions among the basic subshapes occurring in the structure of **s**. Usually, the result should be zero because repetitions of the exact same basic shapes is somewhat unnatural - however, it can happen.

Para testar repetições, use a igualdade "=". Por exemplo, se houver dois círculos iguais (com o mesmo centro e raio) e as restantes forma básicas forem únicas, então o resultado será 2.

9

```
svg : shape -> string
let svg s = ...
```

Converts a shape **s** to a string describing an HTML page that would actually display **s** graphically. The shape should be represented using the SVG format (Scalable Vector Graphics). Therefore, the result should be a string like this: `"<html><body><svg ...> ... </svg></body></html>"`.

You will need to do some research. You can start here.

Note that SVG in HTML use the same coordinate system we are using.

```
partition : shape -> shape list
let partition s = ...
```

Given a shape **s**, determines the maximum number of non-overlapping/non-touching subshapes that can be identified. In the initial example, the result should be a list of four shapes. If we call each element of the partition an **island**, then the initial example is constituted by four islands.

Evaluation and grades

You will submit the file "Shape.ml" via Mooshak.

Around 80% of the grade of your group is automatically assigned by Mooshak. The remaining 20% is assigned manually by the teacher, who will analyze the quality of your code.

A special case: In case of code of extremely bad quality, or code that uses the forbidden imperative mechanisms of OCaml, or code that constantly simulates imperative mechanisms and concepts, a special rule will be used so that the grade will be always below 50%, even if the program works well.

To develop the project, we strongly recommend you use the OCaml interpreter, probably inside the Eclipse IDE.

However, to compile your module, Mooshak will use the following command in Ubuntu 16.04:

```
ocamlc -c Shape.mli Shape.ml
```

After the compilation, Mooshak will test the module in the interpreter like this:

```
$ ocaml
    Objective Caml version 4.02.3
# #load "Shape.cmo";;
# open Shape;;
...
...
```

Please, make a backup of the file "Shape.ml" once a while, because the file can disappear as result of human error or as result of a software/hardware malfunction.

Regras principais

- Produza um ficheiro chamado `Shape.ml`. Nas regras de submissão, a publicar mais tarde, será explicada a forma de submeter no Mooshak.
- O ficheiro "Shape.ml" tem de incluir logo nas primeiras linhas, um comentário inicial contendo: **o nome e número dos alunos que realizaram o projeto**; indicação de quais as partes do trabalho que foram feitas e das que não foram feitas (para facilitar uma correção sem enganos); ainda possivelmente alertando para alguns aspetos da implementação que possam ser menos óbvios para o avaliador.

- O projeto pode ser realizado individualmente ou por grupos de dois alunos. No caso de ser realizado em grupo de dois alunos, será dado um bónus simbólico de 0.5 valores (em todo o caso, a nota final deste projeto nunca poderá ser superior a 20 valores). Um projeto entregue por três ou mais alunos vale zero valores (i.e. grupos com três ou mais alunos são proibidos). Os alunos podem formar grupo com liberdade, independentemente dos turnos em que estejam inscritos.
- Na realização deste projeto é proibido usar os mecanismos imperativos que a linguagem OCaml suporta mas não foram estudados nas aulas.
- Programe as funções recursivas usando o método indutivo. Também pode usar livremente funções de biblioteca, especialmente as disponíveis no módulo List.
- O programa deve ser bem indentado, por forma a ficar bem legível. Além disso, a largura do programa não deve exceder as 80 colunas para poderem ser impressos. Podem haver algumas exceções, muito pontuais. Considera-se que um TAB ocupa quatro posições.

Regras de entrega

- Será ativado um concurso do Mooshak, que servirá para submeter os trabalhos. Os detalhes da forma de fazer a submissão serão divulgados nessa altura. Até lá preocupe-se apenas em escrever um bom programa.
- Depois do prazo limite ainda se aceitam trabalhos atrasados, mas com penalizações na nota. Mais detalhes nas primeiras linhas deste enunciado.

Outras regras

- Apesar de o projeto ser de grupo, cada aluno, a título individual, tem a responsabilidade de responder por todo o projeto. Assim é indispensável que os dois membros de cada grupo programem efetivamente.
- Não se proíbe que alunos de turnos práticos diferentes façam grupo. Isso é apenas desaconselhado.
- Não há inscrição prévia dos grupos e basta que cada trabalho tenha 2 autores identificados.
- A nota máxima do projeto é 20 valores.

Avaliação

O docente responsável pela gestão e pela avaliação deste trabalho é Artur Miguel Dias.

A nota do projeto será em grande parte determinada por meios automáticos, através do Mooshak. Portanto é essencial respeitar a especificação contida neste enunciado, em todos os seus detalhes.

Mas, relativamente a programas que funcionem minimamente, também haverá uma apreciação mais subjetiva da qualidade, tendo em conta aspetos, tais como

- organização,
- clareza e simplicidade das ideias programadas,
- bom uso da linguagem,
- legibilidade do código,
- em alguma medida, eficiência.

Obviamente não é obrigatório fazer o trabalho todo para obter nota positiva. Mas, claro, vale a pena trabalhar para produzir uma solução bastante completa e com a melhor qualidade possível.

Observações

- Os grupos são incentivados a discutir entre si os aspetos gerais do projeto, inclusivamente no fórum. Mas sempre que chega o momento de escrever código concreto, esse tem de ser um esforço interno a cada grupo (trabalhando de forma independente de todos os outros grupos). A escrita de código exige esforço intelectual, mas só com esforço se consegue evoluir.

- O objetivo deste projeto é levar os alunos a praticar. Um aluno que pratique de forma genuína ganha experiência e provavelmente não terá dificuldade em conseguir aprovação nos testes e exames.

- Cuidado com as fraudes. Por exemplo, se alguém dum grupo oferecer o projeto resolvido a um elemento de outro grupo, trata-se duma fraude envolvendo dois grupos. Também se um grupo deixa distraidamente a área aberta e se alguém de outro grupo "rouba" o projeto, então também se considera fraude dos dois grupos. Ainda um terceiro caso: se dois grupos se juntam para fazer o projeto conjuntamente e depois o entregam em duplicado, então também se considera fraude. Em suma, cada grupo é responsável pelo seu projeto e não o pode mostrar ou oferecer, direta ou indiretamente, de propósito ou sem querer, o seu código a outro grupo. Note que é muito melhor ter zero num dos três projetos do que ser logo excluído da cadeira por motivo de fraude.

- Poderá haver discussões dos projetos via Skype, mas será normal os docentes entenderem dispensar a maioria dos alunos da discussão.

Final

Bom trabalho! Esperamos que goste.