enunciat.md - Grip

Pràctica AP2 2019

Aquesta pràctica té dues parts:

Primera part

A la primera part heu d'escriure una classe per tractar polígons convexes i un programa principal amb la calculadora de polígons. Més avall es dóna l'enunciat, en anglès.

Observacions:

- Per a la classe de polígons, trieu la interfície que us sembli més apropiada i còmoda *des del punt de vista de l'usuari*. Trieu també la representació i la implementació el més eficient possible (suposeu que cal tractar polígons amb molts vèrtexs).
- Escriviu diferents jocs de proves que posin de manifest el correcte funcionament i l'eficiència de la vostra solució.
- Documenteu adequadament totes les parts de la vostre feina. Deixeu clares les vostres decisions de disseny.
- Escriviu un fitxer `README.md` per la documentació general, les instruccions d'instal·lació, ètc, i un fitxer `Makefile` per compilar el projecte.
- Assegureu-vos que el vostre codi es compila adequadament, almenys, amb els ordinadors de la facultat.
- Per evitar problemes de còpies, no pengeu el vostre projecte en repositoris públics. Si us cal un repositori GIT, useu GITLAB FIB.

Heu de lliurar la pràctica a través de Peergrade (Learn by giving feedback). Per a fer-ho, aneu a https://app.peergrade.io/join i doneu el codi `GNYBYX`. A continuació, creu un nou compte introduint el vostre nom complet, el vostre correu electrònic oficial (el mateix que teniu pel Jutge) i una contrasenya (recordeu-la!). Lliureu en un fitxer ZIP tots els fitxers necessaris (binaris no!) però tingueu cura de **NO** identificar-ne cap amb el vostre nom o altre informació personal vostra: el vostre lliurament ha de ser completament anònim.

La data límit per lliurar la primera part de la vostra pràctica és el diumenge 31 de març fins a les 23:59.

Segona part

A la segona part de la pràctica haureu de corregir tres pràctiques d'altres companys. Aquesta correcció es farà també a través de Peergrade i implicarà valorar diferents rúbriques que només veureu en aquest punt.

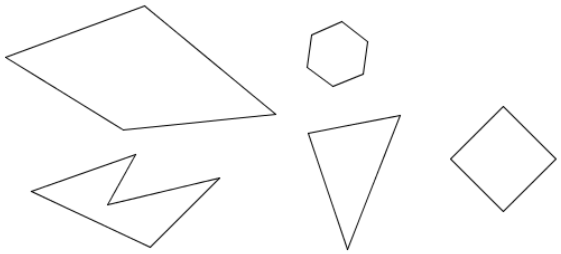
L'avaluació també serà anònima. El sistema calcularà automàticament la nota de cada estudiant i també avisarà als professors de possibles incoherències. Els abusos seran penalitzats. Cada estudiant té el dret de rebutjar la nota rebuda pels seus companys i pot demanar l'avaluació per part d'un professor (qui podrà puntuar a l'alta o a la baixa respecte l'avaluació dels estudiants). Els professors també poden corregir pràctiques "d'ofici" i substituir la nota rebuda pels companys per la del professor.

Podeu començar a corregir les pràctiques dels vostres companys a partir del dimecres 3 d'abril a les 8:00. La data límit per lliurar la segona part de la vostra pràctica és el divendres 26 d'abril fins a les 23:59. No podreu veure les correccions dels vostres companys fins que no hagueu donat les vostres correccions.

Totes les pràctiques s'han de fer en solitari. Els professors utilitzaran programes detectors de plagi. És obligatori corregir les pràctiques dels tres companys assignades pel sistema.

Convex Polygons

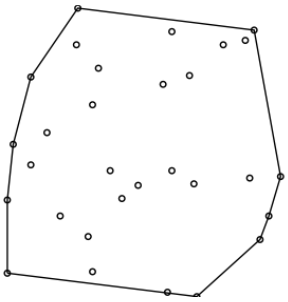
A convex polygon is a simple polygon in which all its interior angles are strictly less than 180°. In the following picture, all polygons are convex, excepting one. Guess which one.



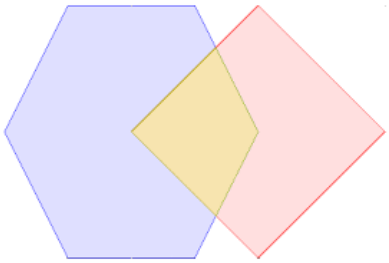
A polygon has *n* vertices and *n* edges. A polygon is said to be *regular* when all edges have the same length. In the picture we have two regular polygons. Can you identify them?

There are several operations that can be done with polygons. Here we mention some examples:

Convex hull: Given a set of points, we can calculate the smallest convex polygon that contains the set.

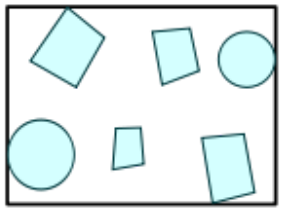


Intersection: The intersection of two convex polygons is a convex polygon, as shown in the figure.



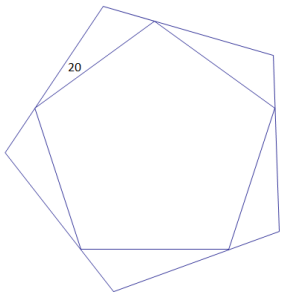
Convex Union: Given two convex polygons, the convex union is the smallest convex polygon that contains both polygons. Equivalently, it is the convex hull of both polygons.

Bounding box: Given a set of convex polygons, find the smallest bounding box that encloses all polygons.



(notice that polygons with a huge number of vertices look like circles in the figure).

Inside: We can check whether a convex polygon is inside another convex polygon.



Information about polygons

Given a polygon, we may want to obtain information about its properties, for example:

- Number of vertices and edges.
- Length of the perimeter.
- Area.
- Is the polygon regular?
- Coordinates of the centroid.

Some particular cases of polygons

For convenience, we will consider some particular cases of polygons (not always recognized as polygons by mathematicians!):

- Empty polygon: a polygon with zero vertices.
- Monogon: a polygon with one vertex (a point).
- Digon: a polygon with two vertices (a segment).

The remaining polygons are more conventional: triangles, quadrilaterals, pentagons, hexagons, etc.

The Class *ConvexPolygon*

The project consists of designing a class to represent and manipulate convex polygons. As part of the project, a simple polygon calculator will have to be created.

Several decisions will have to be taken in the design of the class *ConvexPolygon*:

- Internal representation of a convex polygon.
- Public and private methods.
- Algorithms to perform the required operations efficiently (remember, \(n \log n\) is better than \(n^2\)).
- Specification and documentation of the *Application Programming Interface* (API).
- Set of test examples to check the functionality of the class.

The Polygon calculator

The Polygon calculator must read commands from the standard input and write their answers to the standard output. In some cases, it also should use some files.

Given that the content of `file2.txt` is

```
p2 1 1 0.5 0.1 0 0 1 0
p3 0.1 0.1
```

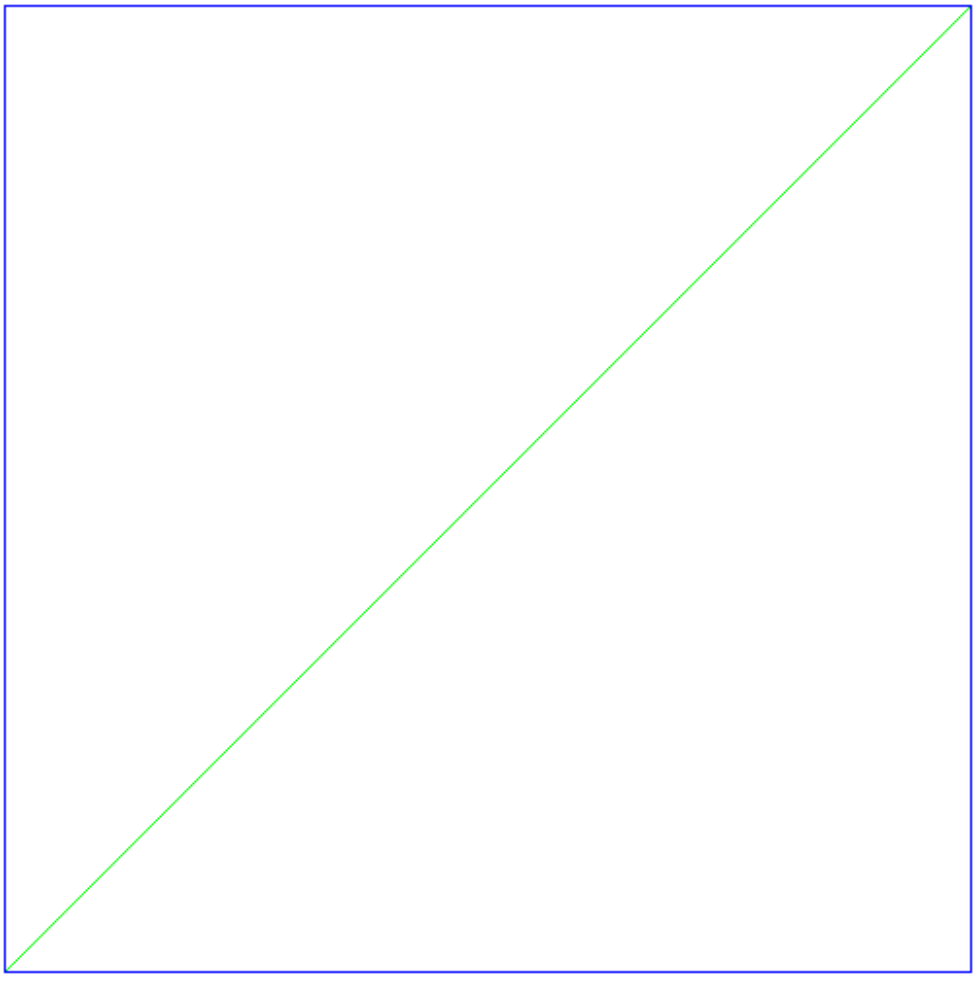
the execution of the script using the calculator on the left should produce the output on the right:

```
# sample script for the polygon calculator #
polygon p1 0 0  0 1  1 1                   ok
print p1                                   p1 0.000 0.000 0.000 1.000 1.000 1.000
area p1                                    0.500
perimeter p1                               3.414
vertices p1                                3
centroid p1                                0.333 0.667
save file1.txt p1                          ok
load file2.txt                             ok
list                                       p1 p2 p3
print p1                                   p1 0.000 0.000 0.000 1.000 1.000 1.000
print p2                                   p2 0.000 0.000 1.000 1.000 1.000 0.000
print p3                                   p3 0.100 0.100
union p3 p1 p2                             ok
print p3                                   p3 0.000 0.000 0.000 1.000 1.000 1.000 1.000 0.000
inside p1 p3                               yes
setcol p1 1 0 0                            ok
setcol p2 0 1 0                            ok
setcol p3 0 0 1                            ok
draw image.png p1 p2 p3                    ok
bbox p4 p1 p2                              ok
print p4                                   p4 0.000 0.000 0.000 1.000 1.000 1.000 1.000 0.000
# some errors                              #
foobar                                     error: unrecognized command
print p5                                   error: undefined identifier
```

Moreover, the content of `file1.txt` will be

```
p1 0.000 0.000 0.000 1.000 1.000 1.000
```

and `image.png` will be



Details of the polygon calculator

The specification of the commands is given bellow. Each command is given in a line and produces exactly one line of output.

Comments

Lines starting with a hash sign (`#`) are comments. Their output is just a hash sign.

Polygon identifiers

All commands include polygon identifiers. These are made by words, such as `p`, `p1`, `p2`, or `pol_gr`.

Points

Points in the commands are given by two pairs of real numbers, in standard notation, to denote the X and Y coordinates. For instance, `0 0` or `3.14 -5.5`. When printed, all real numbers must be formatted with three digits after the decimal dot.

Colors

Colors in the commands are given by three real numbers in [0,1], in standard notation, to denote the RGB color. For instance, `0 0 0` denotes black, `1 0 0` denotes red, and `1 0.64 0` denotes orange.

File names

Filenames in the commands are made up of words, such as `f`, `pol.txt` or `some_file_name.pol`.

The `polygon` command

The `polygon` command associates an identifier with a convex polygon made by a set of zero or more points. If the polygon identifier is new, it will create it. If it already existed, it will overwrite the previous polygon. New polygons are black.

The `print` command

The `print` command prints the name and the vertices of a vertices of a given polygon. The output must only contain the vertices in the convex hull of the polygon, in clockwise order, starting from the vertex will lower X (and the vertex with lower Y in case of ties). They must be printed in a single line, with one space separating each value.

The `area` command

The `area` command prints the area of the given polygon.

The `perimeter` command

The `perimeter` command prints the perimeter of the given polygon.

The `vertices` command

The `vertices` command prints the number of vertices of the convex hull of the given polygon.

The `centroid` command

The `centroid` command prints the centroid of the given polygon.

The `list` command

The `list` command lists all polygon identifiers, lexycographically sorted.

The `save` command

The `save` command saves the given polygons in a file, overwriting it if it already existed. The contents of the file must be the same as in the `print` command, with a polygon per line.

The `load` command

The `load` command loads the polygons stored in a file, in the same way as `polygon`, but retrieving the vertices and identifiers from the file.

The `setcol` command

The `setcol` command associates a color to the given polygon.

The `draw` command

The `draw` command draws a list of polygons in a PNG file, each one with its associated color. The image should be of 500x500 pixels, with white background and the coordinates of the vertices should be scaled to fit in the 498x498 central part of the image, while preserving the original aspect ratio.

The `intersection` command

This command may receive two or three parameters:

- When receiving two parameters `p1`and `p2`, `p1`should be updated to the
- intersection of the original `p1` and `p2`.
- When receiving three parameters `p1`, `p2` and `p3`, `p1`should be updated
- to the intersection of `p2` and `p3`.

Take into account that identifiers may be repeated.

The `union` command

Just as the `intersection` command, but with the convex union of polygons.

The `inside` command

Given two polygons, the `inside` command prints `yes` or `not` to tell whether the first is inside the second or not.

The `bbox` command

The `bbox` command creates a new polygon with the four vertices corresponding to the bounding box of the given polygons.

Commands without answer

As seen in the examples, some commands do not really produce an answer. In this case `ok` must be printed, unless there was some error.

Errors

If any command contains or produces an error, the error must be printed in a line starting with `error:` and the command must be completely ignored (as if it was not given). Possible errors include:

- invalid command
- command with wrong number or type of arguments
- undefined polygon identifier
- wrong format
- ...

Precision

In order to cope with precision issues of float numbers, use an absolute tolerance of `1e-12` when comparing values.

Additional commands

Your project may add new additional commands, provided they are dully documented. Of course, these commands will not be compatible with the test sets of other teams.