Exercise 7
Modules and Types
(16.01.2026)

---

1. Task *A PostScript Module* (10 P)

We want to program a sort of PostScript driver for graphical output as a *module*. That means the output should be directed to a PostScript file, initially implementing only the drawing of lines and circles. The module should have the following properties:

a) It defines variables, such as `scale` for scaling (e.g., `scale = 400.0`) and `psfile_unit` for the file unit (e.g., `psfile_unit = 20`). Additionally, it defines a `TYPE` "point" with the components x-position and y-position.

b) It contains a subroutine for drawing a circle. This subroutine gets a point (see above) and a radius as arguments. The numbers provided are scaled using the "scale" variable. The command to be written to the PS file has the following form:

```
x y r 0 360 arc stroke
```

where `x`, `y`, and `r` are integers.

c) There is a routine for drawing lines, which gets two points (see above) as arguments. The coordinates are again scaled using the `scale` variable. A line from (x_1, y_1) to (x_2, y_2) is created in PostScript with two commands.

```
x_1  y_1  moveto
x_2  y_2  lineto stroke
```

d) A *function* is used for opening the PostScript file. In doing so, the header

```
%!PS-Adobe-1.0
gsave
```

should be written to the file right away. The return value of the function should indicate whether the file was successfully opened.

e) Similarly, a function should be responsible for closing the PostScript file, analogous to the opening process. The PostScript file ends with the following line:

```
showpage grestore
```

The module should now also be utilized. Write a test program that draws a regular polygon with `N` sides, with a small circle at each vertex. For simplicity, you should draw a line from each vertex to all other vertices (see Fig. 1). The i-th vertex of the polygon can be calculated as follows:

```
phi = ( REAL (i) - 0.5 ) * 2.* PI / REAL (N)
point(i)%x = 0.6 + 0.5 * COS (phi)
point(i)%y = 0.6 + 0.5 * SIN (phi)
```

Bonus: Also try to create an appropriate bounding box. This should be exactly the same size as the graphic to be displayed. The bounding box follows the first line and has the following form:

```
%%BoundingBox: xmin ymin xmax ymax
```

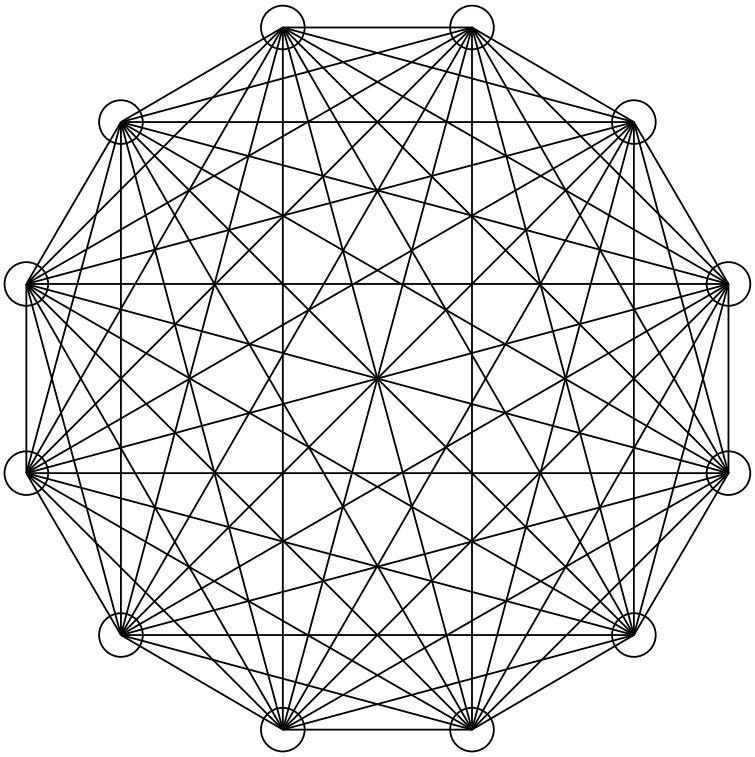

Figure 1: A duodecagon

Hints:

- The line color can be changed with the PostScript command `setrgbcolor`:
  `r g b` `setrgbcolor`
  where r, g, b are the values for the red, green, and blue channel (each $\in [0; 1]$).
- Circles can be filled with a preceding `newpath` and trailing `closepath` and `fill`, e.g.:

```
newpath
   402    358     12  0 360 arc closepath
fill
```