Dr. Andreas Franz Borchert
Institut für Angewandte Informationsverarbeitung

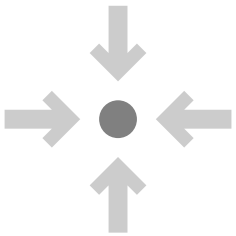
Digitale Typografie (SS 2011)

Abgabetermin: 6. Mai 2011, 8 Uhr

Aufgabe 1

Ziel dieser Aufgabe ist es, erste Übung zu gewinnen mit der Definition von Kurven und der Anwendung von Transformationen in PostScript.

Dazu sollten Sie ein oder zwei einfache geometrische Figuren auswählen und davon mindestens eine mehrfach in ihre Grafik platzieren. So lassen sich beispielsweise vier Pfeile und ein Kreis zu einer Grafik kombinieren, die einen Treffpunkt symbolisieren könnte:



Es steht Ihnen dabei frei, ob Sie das gleiche Logo nachprogrammieren oder Sie einer Ihrer eigenen kreativen Ideen folgen. Es kommt nur auf folgende Punkte an:

- Verpacken Sie jede Ihrer geometrischen Figuren in eine Prozedur, die mit *newpath* beginnt und dann die vollständige Kurve beschreibt, ohne diese jedoch zu zeichnen.
- Jede Zeichenoperation sollte in *gsave* und *grestore* eingebettet werden und mit Hilfe von *translate*, *rotate* und *scale* in der gewünschten Ausrichtung plaziert werden.
- Bestimmen Sie die Bounding-Box Ihrer Grafik und legen Sie (wie im Vorlesungsbeispiel) die beiden Kommentar-Header an, die dafür sorgen, dass die Grafik von *gv* als EPS akzeptiert wird.

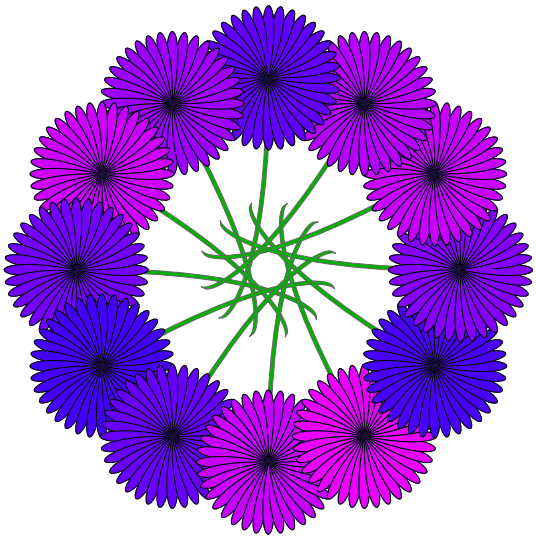
Bitte reichen Sie Ihre Lösung ein mit dem Kommando

```
submit typo 1 symbol.eps
```

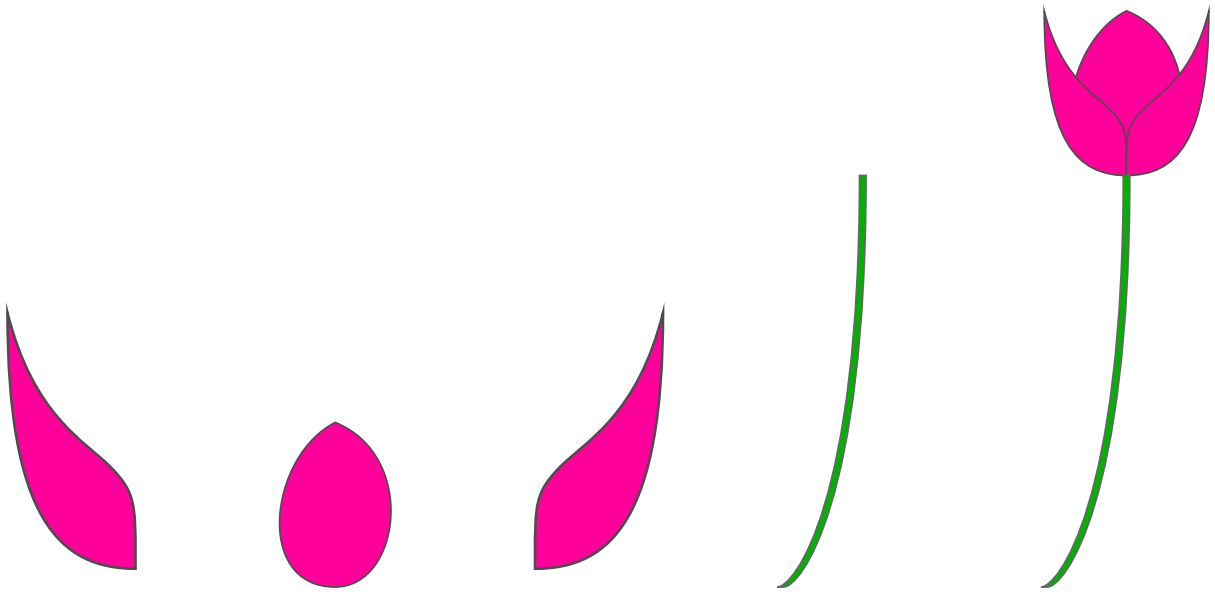
Aufgabe 2

Ziel dieser Aufgabe ist es, Übung zu gewinnen mit der Definition von Bézier-Kurven und den Kontrollstrukturen in PostScript.

Wählen Sie dazu mehrere Kurven aus, die u.a. auch Bézier-Kurven enthalten und die zu einer komplexeren Figur zusammengesetzt werden. Diese Figur ist dann in geeigneter Weise zufällig oder arrangiert vielfach in Ihrer Grafik unterzubringen. Thematisch empfiehlt sich jetzt eine Anlehnung an die Blütenpracht im Mai:



Eine Tulpe aus dem rechten Bild besteht dabei aus drei Blütenblättern und einem Stengel:



Folgende Punkte sind bei Ihrer Lösung zu beachten:

- Für jede einzelne Komponente sollte eine Prozedur definiert werden, die den zugehörigen Pfad definiert.

- Separate Prozeduren sollten dann die einzelnen Pfade ausfüllen und zusammensetzen.
- Wiederum separat ist die vielfache Unterbringung zu lösen.
- Verwenden Sie – soweit es sinnvoll ist – geeignete Parameter und sichern sie diese jeweils in einen lokalen Namensraum (unter Verwendung von *dict*, *begin* und *end*).

Die Lösung ist bitte folgendermaßen einzureichen:

```
submit typo 2 flowers.eps
```

Tipps und Tricks:

Farben können Sie mit dem Operator *setrgbcolor* oder dem Operator *sethsbcolor* setzen, die jeweils drei Gleitkommazahlen zwischen 0 und 1 erwarten. Beispiele:

```
1 0 0 setrgbcolor % liefert eine knallrote Farbe
0.9 1 1 sethsbcolor % liefert die Farbe der Tulpe
```

Grundsätzlich steht Ihnen in PostScript ein Zufallszahlengenerator mit dem Operator *rand* zur Verfügung, der eine ganze Zahl aus dem Bereich 0 bis $2^{31} - 1$ liefert. Sie können diesen einsetzen, um zufällige Positionen oder Farben zu erzeugen. So wurden beispielsweise die Farben im Tulpen-Bild erzeugt:

```
rand 32 mod 220 add % liefert Zahlen aus [220..251]
256 div % bildet dies in [220/256 .. 251/256] ab
1 1 % voll gesaettigt und ausgeleuchtet
sethsbcolor % waehlt diese Farbe nach dem HSB-Farbmodell aus
```

In beiden Beispielen werden die einzelnen Pfade sowohl mit einer Farbe ausgefüllt als auch mit einem Rand versehen, der in einer anderen Farbe ist. Das geht, indem Sie einen Pfad sowohl mit *fill* ausfüllen als auch mit *stroke* ausmalen. Es gibt da nur ein kleines Problem mit einer Folge dieser beiden Kommandos, weil nach *fill* der aktuelle Pfad eliminiert wird, so dass anschließend ein neuer Pfad begonnen werden kann. Das Problem lässt sich mit Hilfe von *gsave* und *grestore* lösen:

```
/Box {
   newpath 0 0 moveto 100 0 lineto 100 100 lineto 0 100 lineto closepath
} def
/PlotBox {
   gsave
      Box
      gsave 1 0 0 setrgbcolor fill grestore % mit roter Farbe ausfuellen
      0.2 setgray 5 setlinewidth stroke % und einen grauen Rahmen hinzuf
   grestore
} def
```

Wenn Sie etwas *n*-mal wiederholen möchten, empfiehlt sich die Verwendung des Operators *repeat*, der als Operanden die Zahl der Durchläufe und etwas Programmtext erwartet. Beispiel:

```
100 {
   gsave
      rand 200 mod rand 200 mod translate
      rand 90 mod rotate
      PlotBox
   grestore
} repeat
```

Eine Schleife, die ganze Zahlen fortlaufend durchiteriert, liefert *for*. Als Operanden werden der Startwert, das Inkrement, der Endwert und der Programmtext erwartet. Bei jedem Schleifendurchlauf wird dann der jeweilige Wert auf den Stack befördert. Beispiel:

```
1 1 10 {
   2 dict begin
   /i exch def
   1 1 10 {
      /j exch def
      gsave
      0.1 0.1 scale
      150 i mul 150 j mul translate PlotBox
      grestore
   } for
   end
} for
```

Viel Erfolg!