16-124-836
Marcel ZAUDER

1.1 PostScript

1.1.1 What kinds of stacks does PostScript manage and what are their roles?

First we have an **Operand Stack** which holds (arbitrary) operands and results of PostScript operators.
Then we have a **Dictionary Stack**, which holds only dictionaries where keys and values may be stores.
An **Execution Stack** holds executable objects like procedures in stages of the execution.
And last we have the **Graphics State Stack** that keeps track of current coordinates.

1.1.2 What is the way of defining a procedure in the PostScript program?

Procedures are defined by binding names to literal or executable objects. For example we can create the following line of code:

```
/square { dup mul } def
```

With this we created a procedure square, which first doublicates the top element of the stack and then multiplies both of the elements. Therefore we can write:

```
3 square
```

and get the result 9.
A procedure calculating $((x + y)/2) \cdot 2$ looks like the following:

```
/calculation { add 2 div 2 mul } def
```

So the whole program can look like the following (example for $x = 3$ and $y = 4$):

```
/procedure { add 2 div 2 mul } def
/sBuf { 20 string } def
/showInt { sBuf cvs show } def
/printCalculation { procedure showInt } def

/Times-Roman findfont 18 scalefont setfont
100 500 moveto
3 4 printCalculation
showpage
```

1.1.3 Procedure for printing 10 random numbers on separate lines

```
/LM 100 def
/FS 18 def
/newline {
    currentpoint exch pop
    FS 2 add sub
    LM exch moveto
} def
/sBuf { 20 string } def
/showInt { sBuf cvs show } def

/Times-Roman findfont 18 scalefont setfont
LM 600 moveto
1 1 10 { rand showInt newline } for
showpage
```

1.2 Catalan Numbers in PostScript

```
/LM 100 def % left margine
/UM 700 def % upper margine
/FS 18 def % font size

/showNum { 20 string cvs show } def

/showCatalan {
    (C \( n = ) show dup showNum ( \) = ) show calculation showNum
} def

/factorial{
    dup 1 lt { pop 1 } { dup 1 sub factorial mul }
    ifelse
} def

/newLine {
    currentpoint exch pop
    FS 2 add sub
    LM exch moveto
} def

/catalan { 0 exch 1 exch { showCatalan newLine } for } def

/calculation {
    dup nominator exch denominator div
} def

/nominator { 2 mul factorial } def

/denominator { dup nplus1fact exch factorial mul } def

/nplus1fact { 1 add factorial } def

/Times-Roman findfont FS scalefont setfont
%Usage: n catalan
LM UM moveto
17 catalan
```

First we create a for loop by adding 0 and 1 in front of n using *exch*. Inside the for loop each **Catalan-Number** is computed by first calculating the nominator and then the denominator. In the end both are used in a division to get the result. These values are then printed with the *showNum* procedure and each value has its individual line, by using the *newLine* procedure.