Project phase 3 — PostScript interpreter core — assigned Thursday 25 September, due Sunday 2 November

Amended 4 October

This is a team exercise. Each team should plan on regular meetings to coordinate progress. During the first week, the team is responsible for developing an outline of the design, including deciding an appropriate Java class and interface structure and writing the Java interfaces. A significant portion of the core interpreter must be designed and written jointly by the team. Later, each member of the team should be assigned specific responsibilities for implementing some of the operators. Each team member is responsible for understanding the structure and the implementation of the *entire* project. Team members will give oral presentations of their work.

In the tradition of 351 project assignments, this is only a first draft and is subject to change (to add further clarifications).

Reading assignment

Read all relevant sections of the *PostScript Language Reference Manual, Second Edition* (PLRM2).

3.1 Task

Write Java classes and interfaces to implement the core functions of a PostScript interpreter.

3.2 Schedule

A revised scanner component is due on Tuesday 30 September. A design document outlining class and interface structure is due on Thursday 2 October *hard copy in class*. A division of labor plan is also due on Thursday 2 October *hard copy in class*. A testing plan including test cases for all operators is due on Monday 6 October. On Tuesday 7 October and Thursday 9 October, instead of standard lectures, the instructor and the teaching assistant will hold conferences with the teams. On Tuesday 28 October and Thursday 30 October, instead of standard lectures, the teaching assistant will hold conferences with the teams.

3.3 Task in detail

3.3.1 Execution model

The basic structure of a PostScript intepreter is given in PLRM2 and has been (and/or will be) elucidated in class discussion. Our execution model is simplified in some ways (see 3.3.4). We do not support all of PostScript data types and operators. In particular, we do not support any graphics-related operators. We also leave out many other operators in this phase.

3.3.2 PostScript operators that must be implemented

Operand Stack Manipulation Operators

- pop
- exch
- dup
- copy (also applicable to arrays and dictionaries)
- index
- roll
- clear
- count
- mark
- cleartomark
- counttomark

Arithmetic and Math Operators

- add
- div
- idiv
- mod
- mul
- sub
- abs

Array Operators

- array
-]
- length (also applicable to dictionaries)
- get (also applicable to dictionaries)
- put (also applicable to dictionaries)
- forall (also applicable to dictionaries; really a control operator)

Dictionary Operators

- dict
- >>
- begin
- end
- def
- currentdict
- countdictstack
- cleardictstack

Relational, Boolean, and Bitwise Operators

- eq
- ne
- ge
- gt
- le
- lt

Control Operators

- exec
- if
- ifelse
- for
- repeat
- loop
- exit

File Operators

- =
- ==
- stack
- pstack

Virtual Memory Operators

- save
- restore

3.3.3 Other PostScript names that must be implemented *Amended 4 October*

- null
- true
- false
- systemdict
- globaldict
- userdict

3.3.4 Where does the program come from?

Unlike a full-fledged PostScript interpreter, your core interpreter only needs to be able to execute code coming from a single input stream, namely a disk file, the name of which is given to the intepreter as its sole command-line argument. Use the scanner from Phase 1 to read in the file and convert it into a list of tokens.

3.3.5 Features to pay attention to

In the scanner:

- correct handling of comments
- correct handling of white space
- correct recognition of numbers, including integers, reals, and radix numbers
- correct handling of integer numbers that exceed the implementation's limit on integer size

- correct handling of strings, including strings in parentheses and hexadecimally encoded strings
- correct handling of *all* string escape sequences (PLRM2, p. 29)

In the design of the interpreter:

- **before committing to *any* decisions about the structure of the interpreter and its VM (memory), make sure you understand how the operators save and restore must work**
- in designing the representation of composite types, realize that, for instance, arrays can share elements (see e.g. the definition of the dictstack operator)

The following types (from the master list on p. 33 of PLRM2) must be implemented:

- boolean
- integer
- mark
- name
- null
- operator
- real
- save
- array
- dictionary
- file

3.3.6 Clarifications of PLRM2

- In general, if you have doubts about what an operator does, i.e., if PLRM2 seems unclear, consult the behavior of GhostScript.
- copy applied to dictionaries: PLRM2 is not explicit, and one might think that dictionary entries are copied according to location (like arrays in PLRM2, and apparently like dictionaries in Level 1 implementations). Instead, all key-value pairs from one dictionary (*dict1*) are dumped into the other (*dict2*). This is the behavior of GhostScript.
- *Amended 4 October* Implementation limits (Appendix B of PLRM2): try to avoid imposing *any* implementation limits.

3.3.7 Deviations from PLRM2

- (inherited from the scanner phase) You do not need to implement the standard PostScript behavior when errors in the input are discovered (such as PostScript limitcheck and syntaxerror errors. Instead, you should declare and raise corresponding Java exceptions. No specific recovery mechanism needs to be implemented. When these errors are raised, a suitable error message must be printed, and then the interpreter may quit.
- This simplified treatment of errors (mapping to Java exceptions) applies to all other PostScript errors as well.
- The operators == and pstack must print objects in the same format as GhostScript does. On the other hand the operators = and stack may print objects in any format whatsoever, but it is recommended that they should print a very detailed, implementation-specific, representation, so that they may be used for debugging the interpreter. For instance, when = is applied to an array, it is useful to print out not only the values of the elements, but also the memory locations where they are stored.
- Only one memory, namely local VM, is supported (see PLRM2, Section 3.7). All heap objects are allocated in local VM. For compatibility, we still have a global dictionary, but it is allocated in local VM and remains empty. (Thus, the initial state of the interpreter includes three dictionaries on the dictionary stack: system, global, and user.)
- Garbage collection (Section 3.7.4) does not need to be implemented.
- *Amended 4 October* The access attribute (Section 3.3.2) does not need to be implemented. Thus, the only object atttribute is the literal/executable attribute.

3.3.8 Features in PLRM2 that need not be implemented

- (inherited from the scanner phase) binary token and binary object sequence encoding (see Section 3.2 and Section 3.12): only the ASCII encoding (Section 3.2.2) needs to be implemented
- (inherited from the scanner phase) ASCII base-85 strings
- all operators not listed in 3.3.2 above
- the following types (from the master list on p. 33 of PLRM2) need not be implemented:
  - fontID
  - condition
  - gstate
  - lock
  - packedarray
  - string
- graphics (to be tackled in a later phase): only reserve placeholders in the state of the interpreter for the current graphics state and for the graphics state stack

- any features having to do with *Display PostScript*
- features having to do with print servers, print job control, etc.

3.4 Hints on the Java implementation

You should reuse your Java representation for PostScript integers, reals, names, and tokens from Phase 1.

How to turn in

There are several intermediate deadlines for different project components, so this applies to each of them.

Turn in your code by running

˜barrick/handin your-file

on a regular UNM CS machine.

You should use whatever filename is appropriate in place of your-file. You can put multiple files on the command line, or even directories. Directories will have their entire contents handed in, so please be sure to clean out any cruft.