CS 201 - Data Structures and Algorithms II: Assignment #1

Muhammad Mobeen Movania (L1),
Syeda Saleha Raza (L2),
Faisal Alvi (L3, L4),
Abdullah Zafar (L5).

Due on February 10, 2024, 11.59pm

Student 1 Name, ID **Student 2 Name, ID**

Instructions

This assignment document consists of two problems.

- Problem 1 is a theoretical question which requires analysis. It should be completed and submitted within this document. This problem is worth 20 points.
- Problem 2 is a programming based question which requires implementation. It must be submitted as a zipped archive containing:

  1. your programs (the language of implementation in C++),
  2. the input files provided to you, and
  3. the output files generated by your program.

  The solution to program 2 will be graded for correctness and structure by testing the output against test input files. This problem is worth 40 points.

Problem 1

(20 points) [**Amortized Analysis**] Let us define a `stop-min-stack` with a stack as the underlying data structure, which stores a sequence of items and has the following operations:

1. `push(`x`)` adds the item x to the top of the underlying stack.
2. `peek()` returns the item at the top of the underlying stack without removing it.
3. `stop-min-pop()` pops a local minimum from the existing items in the stack. This operation works as follows:
   (a) removes an existing item from the stack using `pop()`, (if the stack was empty before the `pop()`, NULL is returned; if the stack is empty as a result of the `pop()`, the popped item is returned)
   (b) tests whether the popped item is greater than or equal to the current item at the top of the underlying stack using `peek()`.
   (c) If yes, the recently popped item is discarded.
   (d) Steps (a) - (c) are repeated as a loop until the recently popped item is less than the topmost item in the stack or if the underlying stack is empty.
   (e) Finally, the recently popped item is returned.

For example if the stack contains the items 4, 3, 2, 9, 5, then after one application of `stop-min-pop()`, the item 2 will be returned and the stack will have the elements 9, 5 as shown below:

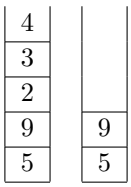

Table 1: `Stop-min-stack` before and after one application of `stop-min-pop()`

(a) (5 points) Using an underlying Stack which has only `push`, `pop` and `peek` as its operations with constant time complexity, write pseudocode for the operation `stop-min-pop()`. What is the time complexity of `stop-min-pop()` in the worst case for a stack of size n?

(b) (5 points) Give an argument using worst case analysis that if `stop-min-pop()` is applied n times to an arbitrary stack of size n, the worst case time complexity can be $O(n^2)$, thereby giving a cost of $O(n)$ per operation.

(c) (10 points) Using amortized analysis, show that the amortized cost of applying `stop-min-pop()` n times to an arbitrary stack of size n is actually $O(1)$ per operation.

Problem 2

(40 points) [**Simplifying Postscript files by evaluating stack based expressions**]
Postscript is a programming language that has been widely used for publishing documents. It is a stack based language that can be used to construct very simple to complex graphics as well as text. Modern documents in PDF (portable document format) use postscript as part of the document structure for rendering graphics. Some examples of postscript graphics are given below:

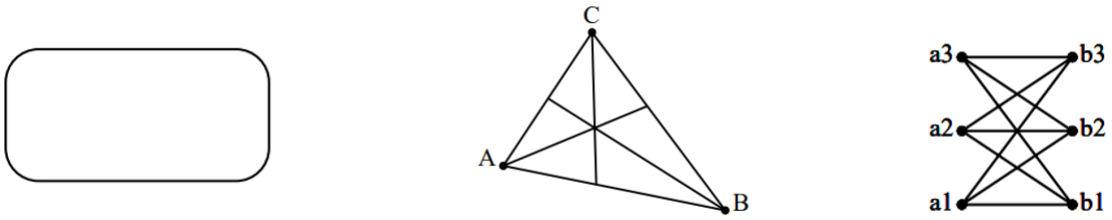

Figure 1: Examples of Postscript Graphics

A postscript file (or encapsulated postscript file) has an extension .ps (or .eps) and can be opened and edited in a text editor. The drawing canvas has coordinates starting from (0, 0) at the bottom left corner of the page with coordinates increasing in the positive X and Y directions. As an example, the following postscript code on the left displays the image on the right.

FILE IMAGE

ch1fig.1.eps

File Edit View

```
newpath
 1  1 moveto
91  1 lineto
91 91 lineto
 1 91 lineto
 1  1 lineto
stroke
showpage
```

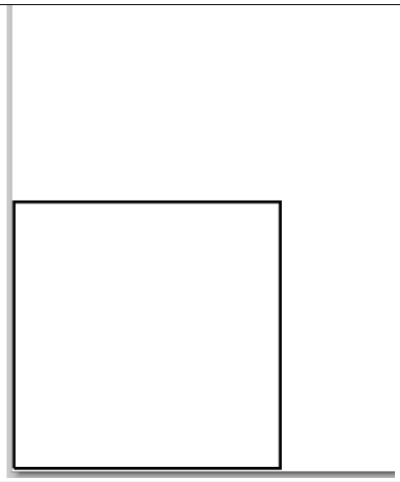

Figure 2: Postscript Code (Left) with Example (Right)

For a more complete reference to postscript you may consult the following tutorial: Learn Postscript by Doing (`https://staff.science.uva.nl/a.j.p.heck/Courses/Mastercourse2005/tutorial.pdf`)

To display a postscript file, several programs can be used both online as well as offline. We use Inkscape (`https://inkscape.org/`) a free and open source vector graphics software to render images in this document.

Stacks: Postscript uses a stack for all its computations including mathematical as well as stack based manipulation. A list of stack based operators for postscript are shown in the following tables:

Arithmetic and Mathematical Operators

Arguments	*Operator*	*Left on Stack; Side Effects*
$x\ y$	`add`	$x + y$
$x\ y$	`sub`	$x - y$
$x\ y$	`mul`	xy
$x\ y$	`div`	x/y
$x\ y$	`idiv`	the integral part of x/y
$x\ y$	`mod`	the remainder of x after division by y
x	`abs`	the absolute value of x
x	`neg`	$-x$
x	`ceiling`	the integer just above x
x	`floor`	the integer just below x
x	`round`	x rounded to nearest integer
x	`truncate`	x with fractional part chopped off
x	`sqrt`	square root of non-negative x
$y\ x$	`atan`	the polar argument of the point (x, y), in degrees between 0 and 360
x	`cos`	$\cos x$ (x in degrees)
x	`sin`	$\sin x$ (x in degrees)
$x\ y$	`exp`	x^y
x	`ln`	$\ln x$
x	`log`	$\log x$ (base 10)
	`rand`	a random integer in the range 0 to $2^{31} - 1$

Figure 3: Arithmetic and Mathematical Operators

Operand Stack Manipulation Operators

Operator	*Meaning*	*Effects on Stack*
`clear`	remove all stack items	$\vdash obj_1 \cdots obj_n$ `clear` $\Longrightarrow \vdash$
`cleartomark`	remove all stack items down through *mark*	*mark* $obj_1 \cdots obj_n$ `cleartomark` $\Longrightarrow -$
`copy`	duplicate top n stack items	$obj_1 \cdots obj_n\ n$ `copy` $\Longrightarrow$ $obj_1 \cdots obj_n\ obj_1 \cdots obj_n$
`count`	count the stack items	$\vdash obj_1 \cdots obj_n$ `count` $\Longrightarrow \vdash obj_1 \cdots obj_n\ n$
`counttomark`	count all stack items down through *mark*	*mark* $obj_1 \cdots obj_n$ `counttomark` $\Longrightarrow$ *mark* $obj_1 \cdots obj_n\ n$
`dup`	duplicate the top stack item	*obj* `dup` $\Longrightarrow$ *obj obj*
`exch`	exchange top two stack items	$obj_1\ obj_2$ `exch` $\Longrightarrow obj_2\ obj_1$
`index`	push a copy of the n-th stack item on the stack	$obj_n \cdots obj_0\ n$ `index` $\Longrightarrow$ $obj_n \cdots obj_0\ obj_n$
`mark`	push mark on stack	`mark` $\Longrightarrow$ *mark*
`pop`	remove the top stack item	*obj* `pop` $\Longrightarrow -$
`roll`	roll n stack items up j times	$obj_{n-1} \cdots obj_0\ n\ j$ `roll` $\Longrightarrow$ $obj_{(j-1) \bmod n} \cdots obj_0\ obj_{n-1} \cdots obj_{j \bmod n}$

Figure 4: Stack Manipulation Operators

For the purpose of this assignment, we will be using the following operators only:

`add, sub, mul, div, sin, cos, mod, exp, sqrt, dup, exch, roll`.

Additionally, you may use any other operators that may be required to implement the above list of operators.

Problem 2 continued on next page...

Variable or Procedure Declaration: You can declare a variable (or a procedure) in postscript using a "/" preceding the variable (or procedure) name, followed by the variable (or procedure) definition, and ending with a "def" as shown in the following figure:

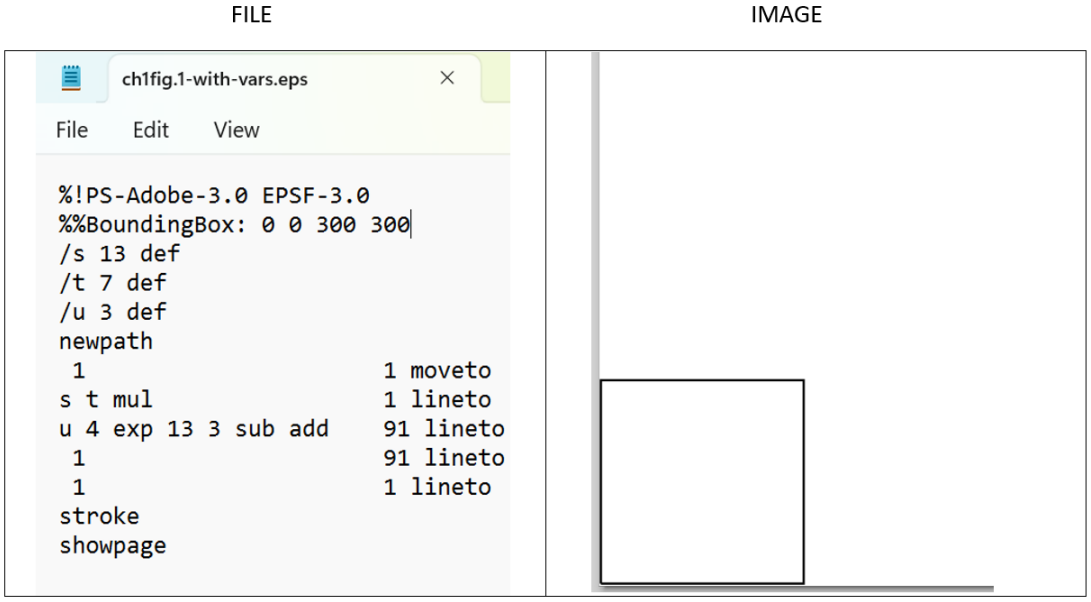

Figure 5: Variable Declarations: s, t and u are variables

Note that this file renders the same square as the previous example. However, here we declare three variables s, t and u. Furthermore,

- `s t mul` evaluates to `13 7 mul` on the stack $\Rightarrow 13 \cdot 7 = 91$
- `u 4 exp 13 3 sub add` evaluates to `3 4 exp 13 3 sub add` $\Rightarrow 3^4 + 13 - 3 = 81 + 10 = 91$, so a simplified version of the file will be:

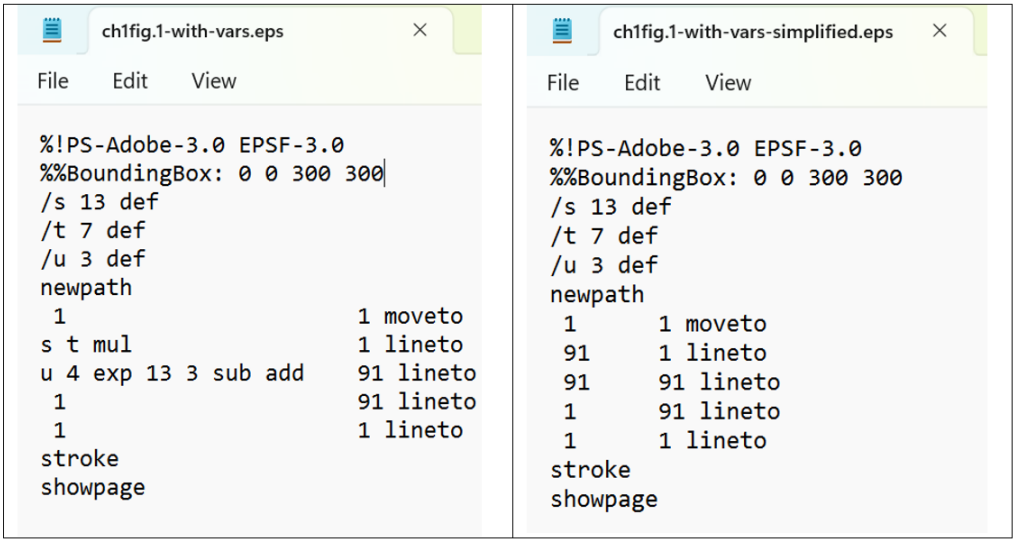

Figure 6: Original File Vs Simplified File with Variable declaration

The following figure gives a procedure definition "/dir" with the original file, simplified file and the rendered image. Note that the procedure definition is enclosed in curly brackets (braces).

ORIGINAL FILE

ch3ex16-as-is.eps

File Edit View

```
%!PS-Adobe-3.0 EPSF-3.0
%%BoundingBox: 0 0 57 60
%%BeginProlog
/dir {dup cos exch sin} def
/s 28.34645 def
%%EndProlog
s s scale
1 s div setlinewidth
0.9 1.05 translate
newpath
1 0 moveto
 72 dir lineto
144 dir lineto
216 dir lineto
288 dir lineto
closepath
stroke
showpage
%%EOF
```

SIMPLIFIED FILE

ch3ex16-as-is.eps

File Edit View

```
%!PS-Adobe-3.0 EPSF-3.0
%%BoundingBox: 0 0 57 60
%%BeginProlog
/dir {dup cos exch sin} def
/s 28.34645 def
%%EndProlog
28.34645 28.34645 scale
0.0352777839 setlinewidth
0.9 1.05 translate
newpath
1 0 moveto
0.309017 0.95105654 lineto
-0.809017 0.587785244 lineto
-0.809017 -0.587785244 lineto
0.309017 -0.95105654 lineto
closepath
stroke
showpage
%%EOF
```

RENDERED IMAGE

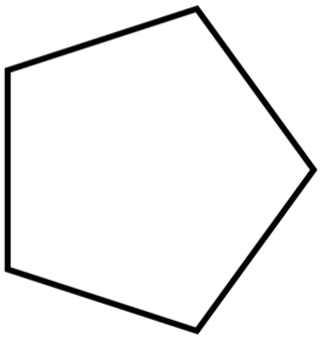

Figure 7: Original File, Simplified File and And Rendered Image

Objective: Given a postscript file with operations from the tables (Arithmetic and Manipulation Operators), generate a simplified postscript file that does not contain any of the stack based operators – instead it contains the results of evaluations of these operators.

Problem Description: Design and implement a class `PostScriptFileSimplifier` in C++ that:

- (5 points) reads in a postscript file into an appropriate data structure of your choice,
- (10 points) replaces all variable and procedure names with their simplified definitions in the data structure. This can be achieved by storing and utilizing information about variables and procedures along with the required functions in another data structure,
- (20 points) has a function to simplify the postscript file by evaluating all stack based operations and removing these from the actual postscript file. You may declare a class `StackPostscript` that uses a Stack data structure and implements various stack based operations as shown in Figures 3 and 4,
- (5 points) writes the simplified postscript file as `<originalfile>-simplified.ps` (or .eps).

We are providing you with two folders: `input` containing input files and `output` containing the corresponding simplified output files. Your program should read in every input file in the input folder and produce the corresponding output files in a folder called "test-output".

Further submission guidelines will be provided to you later.

Due date: Saturday, 10 February 2024, 11.59pm.