جامعة بيرزيت

BIRZEIT UNIVERSITY

FACULTY OF ENGINEERING

ELECTRICAL AND COMPUTER ENGINEERING DEPARTMENT

LINUX LABORATORY : ENCS3130

Instructor: Dr. Aziz Qaroush

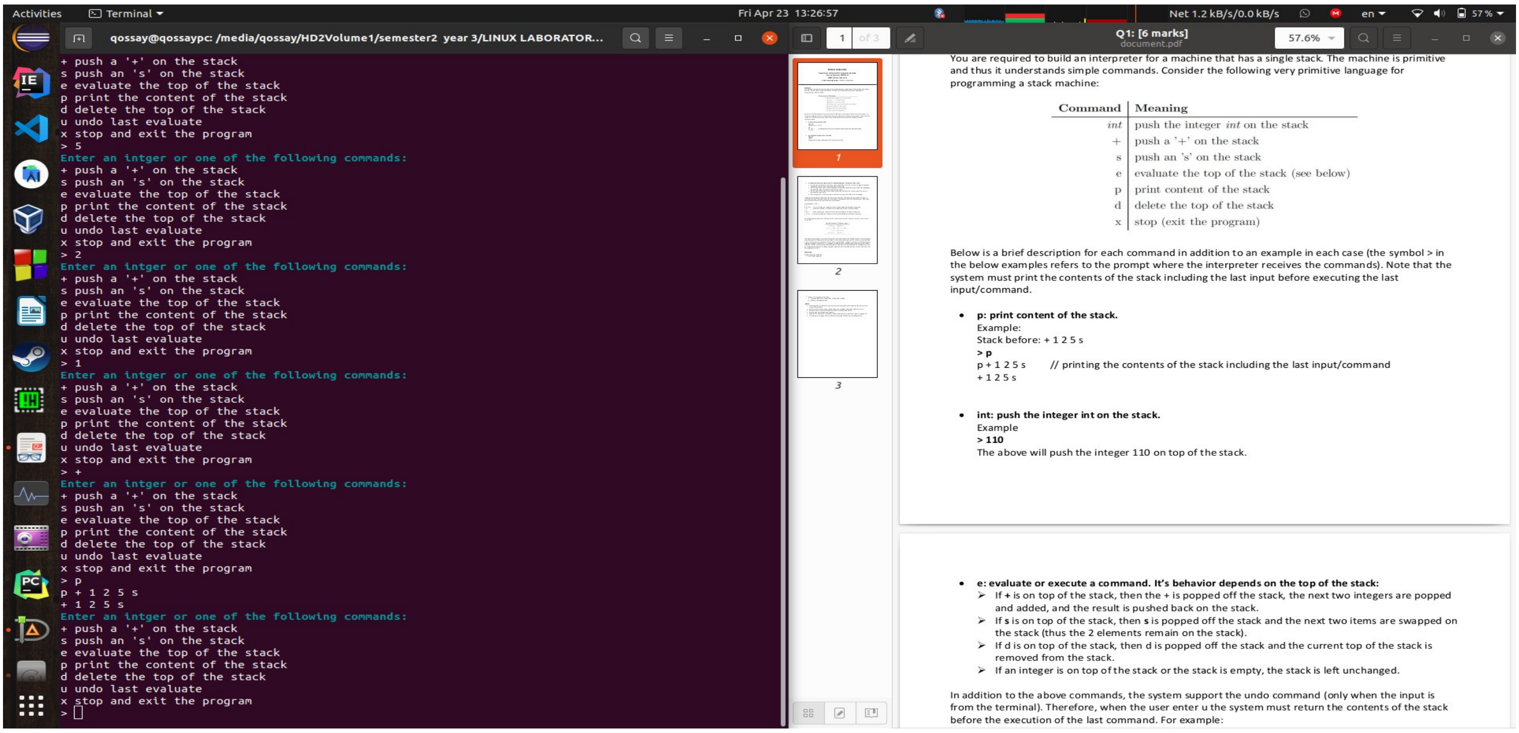
Qossay ZeinElddin: 1180235

Section : 2

date 22/4/2021

first i implemente emty array to add elemants in
second write to founction pop to remove frome stack and pouh to add to stack
and stackTop function tp get the last elemant enter to stack and stackSize fun to get the size of arry start from1 then stackPrint fun thant print elemant in array as stack from last elemant in arrray to index 0
then we enter while loup and print the lest and then do all function in project

Examples :-



qossay@qossaypc: /media/qossay/HD2Volume1/semester2 year 3/LINUX LABORATOR...

```
+ push a '+' on the stack
s push an 's' on the stack
e evaluate the top of the stack
p print the content of the stack
d delete the top of the stack
u undo last evaluate
x stop and exit the program
> 5
Enter an intger or one of the following commands:
+ push a '+' on the stack
s push an 's' on the stack
e evaluate the top of the stack
p print the content of the stack
d delete the top of the stack
u undo last evaluate
x stop and exit the program
> 2
Enter an intger or one of the following commands:
+ push a '+' on the stack
s push an 's' on the stack
e evaluate the top of the stack
p print the content of the stack
d delete the top of the stack
u undo last evaluate
x stop and exit the program
> 1
Enter an intger or one of the following commands:
+ push a '+' on the stack
s push an 's' on the stack
e evaluate the top of the stack
p print the content of the stack
d delete the top of the stack
u undo last evaluate
x stop and exit the program
> +
Enter an intger or one of the following commands:
+ push a '+' on the stack
s push an 's' on the stack
e evaluate the top of the stack
p print the content of the stack
d delete the top of the stack
u undo last evaluate
x stop and exit the program
> p
p + 1 2 5 s
+ 1 2 5 s
Enter an intger or one of the following commands:
+ push a '+' on the stack
s push an 's' on the stack
e evaluate the top of the stack
p print the content of the stack
d delete the top of the stack
u undo last evaluate
x stop and exit the program
> 
```

You are required to build an interpreter for a machine that has a single stack. The machine is primitive and thus it understands simple commands. Consider the following very primitive language for programming a stack machine:

Command	Meaning
int	push the integer *int* on the stack
+	push a '+' on the stack
s	push an 's' on the stack
e	evaluate the top of the stack (see below)
p	print content of the stack
d	delete the top of the stack
x	stop (exit the program)

Below is a brief description for each command in addition to an example in each case (the symbol > in the below examples refers to the prompt where the interpreter receives the commands). Note that the system must print the contents of the stack including the last input before executing the last input/command.

- **p: print content of the stack.**
  Example:
  Stack before: + 1 2 5 s
  > p
  p + 1 2 5 s // printing the contents of the stack including the last input/command
  + 1 2 5 s

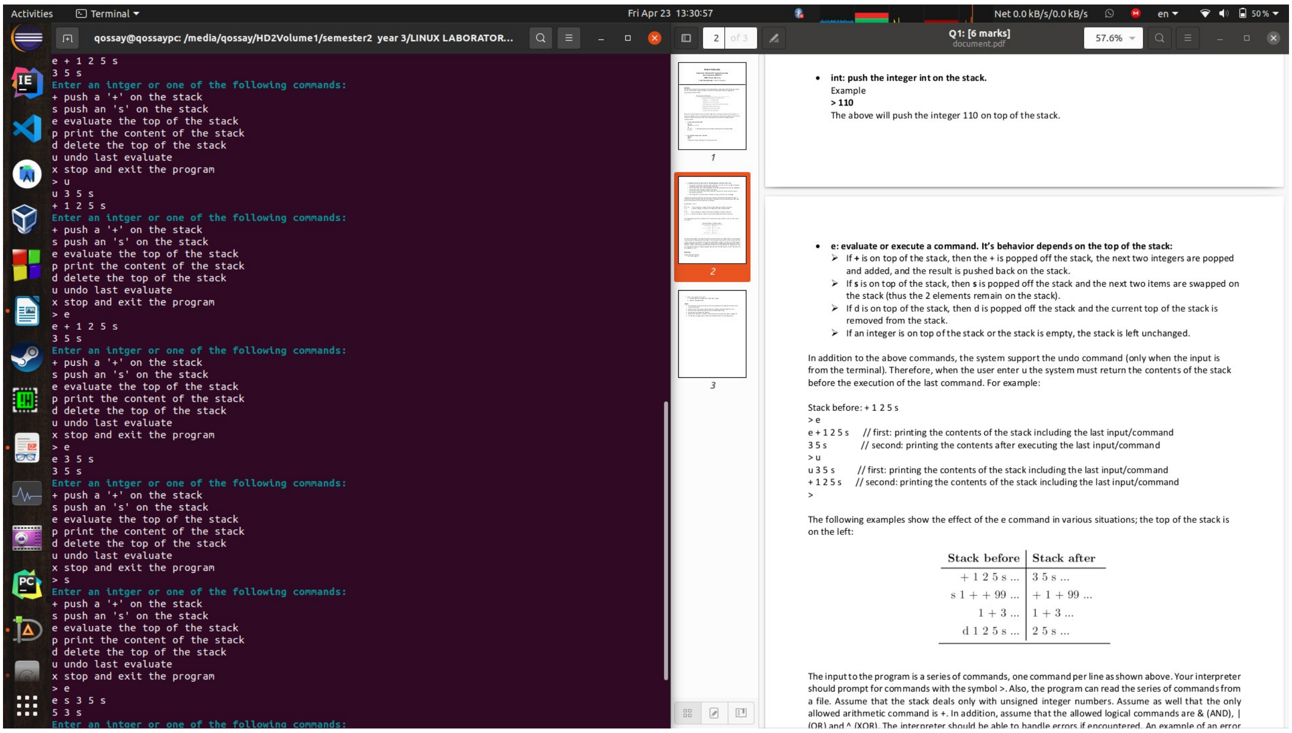
- **int: push the integer int on the stack.**
  Example
  > 110
  The above will push the integer 110 on top of the stack.

- **e: evaluate or execute a command. It's behavior depends on the top of the stack:**
  - If **+** is on top of the stack, then the + is popped off the stack, the next two integers are popped and added, and the result is pushed back on the stack.
  - If **s** is on top of the stack, then **s** is popped off the stack and the next two items are swapped on the stack (thus the 2 elements remain on the stack).
  - If d is on top of the stack, then d is popped off the stack and the current top of the stack is removed from the stack.
  - If an integer is on top of the stack or the stack is empty, the stack is left unchanged.

In addition to the above commands, the system support the undo command (only when the input is from the terminal). Therefore, when the user enter u the system must return the contents of the stack before the execution of the last command. For example:

.

In this exmple we insert into the stack s -> 5 -> 2 -> 1 -> +
then we print the stack ass enter “p”



qossay@qossaypc: /media/qossay/HD2Volume1/semester2 year 3/LINUX LABORATOR...

```
e + 1 2 5 s
3 5 s
Enter an intger or one of the following commands:
+ push a '+' on the stack
s push an 's' on the stack
e evaluate the top of the stack
p print the content of the stack
d delete the top of the stack
u undo last evaluate
x stop and exit the program
> u
u 3 5 s
+ 1 2 5 s
Enter an intger or one of the following commands:
+ push a '+' on the stack
s push an 's' on the stack
e evaluate the top of the stack
p print the content of the stack
d delete the top of the stack
u undo last evaluate
x stop and exit the program
> e
e + 1 2 5 s
3 5 s
Enter an intger or one of the following commands:
+ push a '+' on the stack
s push an 's' on the stack
e evaluate the top of the stack
p print the content of the stack
d delete the top of the stack
u undo last evaluate
x stop and exit the program
> e
e 3 5 s
3 5 s
Enter an intger or one of the following commands:
+ push a '+' on the stack
s push an 's' on the stack
e evaluate the top of the stack
p print the content of the stack
d delete the top of the stack
u undo last evaluate
x stop and exit the program
> s
Enter an intger or one of the following commands:
+ push a '+' on the stack
s push an 's' on the stack
e evaluate the top of the stack
p print the content of the stack
d delete the top of the stack
u undo last evaluate
x stop and exit the program
> e
e s 3 5 s
5 3 s
Enter an intger or one of the following commands:
```

- **int: push the integer int on the stack.**
  Example
  > 110
  The above will push the integer 110 on top of the stack.

- **e: evaluate or execute a command. It's behavior depends on the top of the stack:**
  - If **+** is on top of the stack, then the + is popped off the stack, the next two integers are popped and added, and the result is pushed back on the stack.
  - If **s** is on top of the stack, then **s** is popped off the stack and the next two items are swapped on the stack (thus the 2 elements remain on the stack).
  - If d is on top of the stack, then d is popped off the stack and the current top of the stack is removed from the stack.
  - If an integer is on top of the stack or the stack is empty, the stack is left unchanged.

In addition to the above commands, the system support the undo command (only when the input is from the terminal). Therefore, when the user enter u the system must return the contents of the stack before the execution of the last command. For example:

```
Stack before: + 1 2 5 s
> e
e + 1 2 5 s   // first: printing the contents of the stack including the last input/command
3 5 s         // second: printing the contents after executing the last input/command
> u
u 3 5 s       // first: printing the contents of the stack including the last input/command
+ 1 2 5 s     // second: printing the contents of the stack including the last input/command
>
```

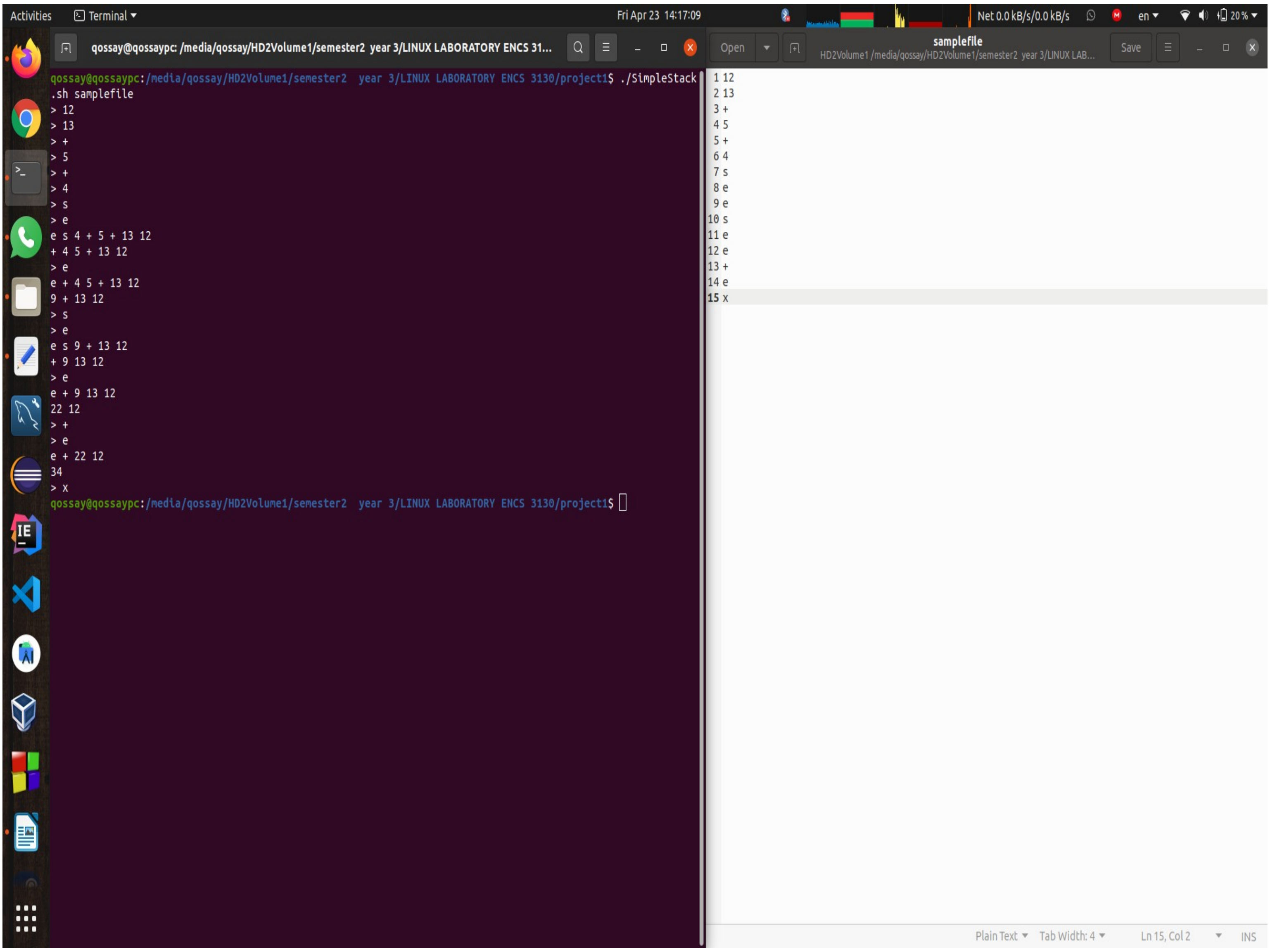
The following examples show the effect of the e command in various situations; the top of the stack is on the left:

Stack before	Stack after
+ 1 2 5 s ...	3 5 s ...
s 1 + + 99 ...	+ 1 + 99 ...
1 + 3 ...	1 + 3 ...
d 1 2 5 s ...	2 5 s ...

The input to the program is a series of commands, one command per line as shown above. Your interpreter should prompt for commands with the symbol >. Also, the program can read the series of commands from a file. Assume that the stack deals only with unsigned integer numbers. Assume as well that the only allowed arithmetic command is +. In addition, assume that the allowed logical commands are & (AND), | (OR) and ^ (XOR). The interpreter should be able to handle errors if encountered. An example of an error

then we enter e to do the function + which add 1and 2 then ths stack become 3 5 s
then we enter e but nothing happen becouse no function there just number so we enter s
the stack now s35s and then enter e to do the function s so thant the stack now 5 -> 3 ->   s

another example from file (the file is with code substution)


Activities
Terminal
Fri Apr 23 14:17:09
Net 0.0 kB/s/0.0 kB/s
en
20 %
qossay@qossaypc: /media/qossay/HD2Volume1/semester2 year 3/LINUX LABORATORY ENCS 31...
Open
samplefile
HD2Volume1 /media/qossay/HD2Volume1/semester2 year 3/LINUX LAB...
Save
Plain Text
Tab Width: 4
Ln 15, Col 2
INS

```
qossay@qossaypc:/media/qossay/HD2Volume1/semester2  year 3/LINUX LABORATORY ENCS 3130/project1$ ./SimpleStack
.sh samplefile
> 12
> 13
> +
> 5
> +
> 4
> s
> e
e s 4 + 5 + 13 12
+ 4 5 + 13 12
> e
e + 4 5 + 13 12
9 + 13 12
> s
> e
e s 9 + 13 12
+ 9 13 12
> e
e + 9 13 12
22 12
> +
> e
e + 22 12
34
> x
qossay@qossaypc:/media/qossay/HD2Volume1/semester2  year 3/LINUX LABORATORY ENCS 3130/project1$ 
```

```
12
13
+
5
+
4
s
e
e
s
e
e
+
e
x
```

we read to the stack from file 12 > 13 >+ > 5 >+>4 >s >e (when read “e” do the funcrion)
so s swapp stack now 12 >13>+>5> 4 >+ then read e so stgack now do fun’+’
12 > 13 > + > 9 then read s then read e so that we will do fun’s’
stack now 12 >14>9>+ after thant read e so do fun ‘+’ stack now 12 > 22 after that read + after that read e so do ‘+’ stack now 34 then read x so exit