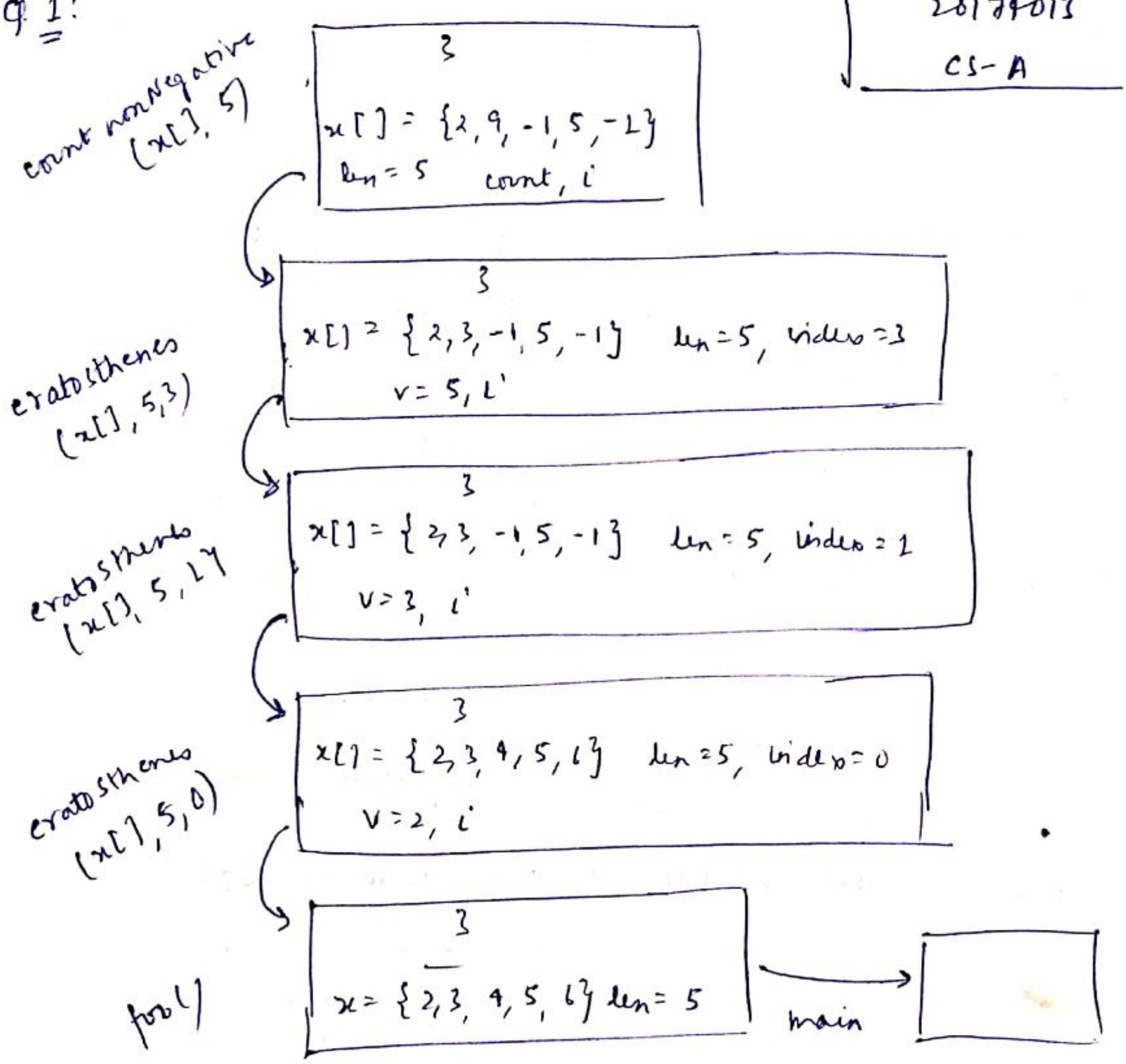
Assign-5 Compiler Construction 7/6/20

Niket Keshari
20174013
CS-A

Q. 1:


count nonNegative (x[], 5)
3
x[] = {2, 9, -1, 5, -2}
len = 5 count, i
eratosthenes (x[], 5, 3)
3
x[] = {2, 3, -1, 5, -1} len = 5, index = 3
v = 5, i
eratosthenes (x[], 5, 1)
3
x[] = {2, 3, -1, 5, -1} len = 5, index = 1
v = 3, i
eratosthenes (x[], 5, 0)
3
x[] = {2, 3, 4, 5, 6} len = 5, index = 0
v = 2, i
foo()
3
x = {2, 3, 4, 5, 6} len = 5
main

2) The aim of this question is to show how compilers handle calling-by-value & calling-by-reference.

⇒ Code for the function is generated once & should be able to retrieve the value passed argument regardless of what is happened before. Other calls to each function might also be present & should work with same code.

→ passed by value - value should be used in activation record

→ " " reference - its value should be retrievably by single deref.

⇒ if foo uses call by value:

```
lw $a0, OFFSET($FP)
```

⇒ if foo uses call by reference:

```
lw $t0, OFFSET($FP)
lw $a0, 0($t0)
```

⇒ The content of argument slots of the activation records will be:-

① foo uses call by value & bar uses call by value, both slots will contain actual values (i.e the numeric representation of 6)

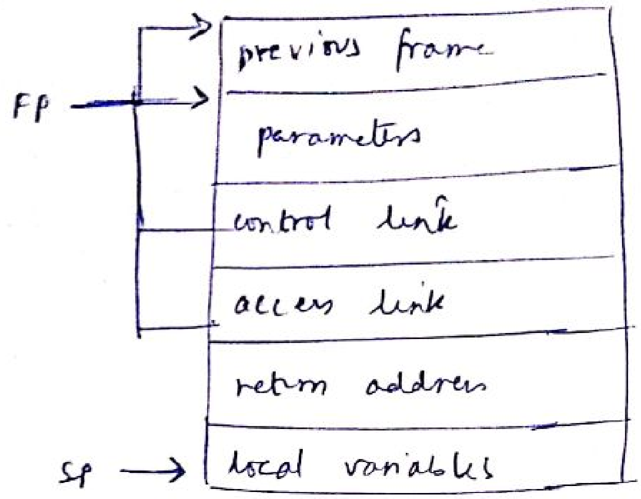
② foo uses call by value & bar uses call by reference, the slot for call on line 13 contains the address of variable c, whereas the slot for call on line 7 contains actual values retrieved from that address before call (ie 6)

③ <u>foo</u>: call by reference; <u>bar</u>: call by value -
The slot for call on line 13 contains value of the variable c at the time of call (ie 6) whereas the slot for call on line 7 contains address of the previous slot

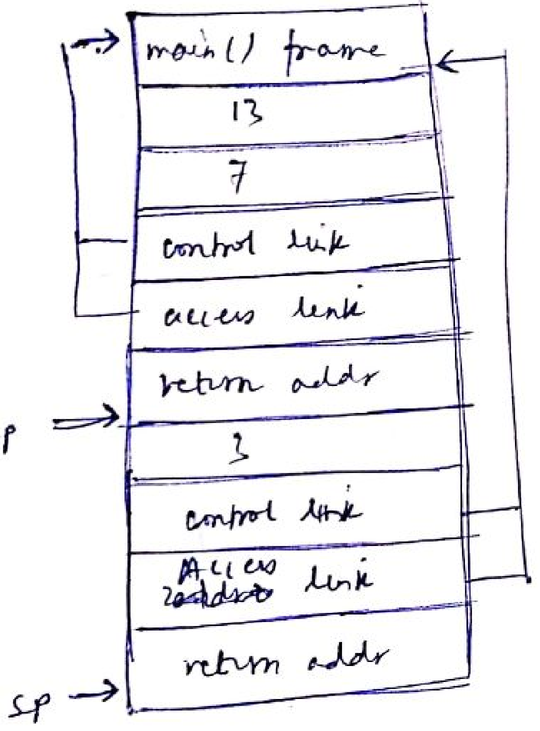
④ <u>foo</u>: call by reference; <u>bar</u>: call by reference -
line 13 contains address of variable c & same address is copied in slot for call on line 7. The compiler knows that f^n recieved an argument by reference & has to pass it on by reference, therefore it copies the original reference.

<u>Q.3</u>: <u>Return value of this call</u>:- spam (7,13) = 3 * 7 mod 13 = 8

<u>Stack</u>: The general idea for a stack layout that supports nested f^n definitions would look like:-



following this stack idea, the stack right before return of the call:-



⇒ x is retrieved directly from by accessing the function's parameter inside function frame:

```
lw $t0, -4($fp)
```

⇒ a is retrieved using access link in the fn. frame as it is defined in a higher scope

```
lw, $t0, -c($fp)
lw, $t0, -8($t0)
```

⇒ This code assumes that values have size of a word

Q.4:

Algebric optimization

a := b+c
z := a*a
x := 0
y := b+c
w := y*y
u := x+3
v := u+w

→ Common sub-expression elim..

a := b+c
z := a*a
x := 0
y := a
w := y*y
u := x+3
v := u+w

→ copy propogation:-

```
a := b + c
z := a * a
x := 0
y := a
w := a * a
u := 0 + 3
v := u + w
```

→ constant folding:-

```
a := b + c
z := a * a
x := 0
y := a
w := a * a
u := 3
v := u + w
```

→ Dead code elimination:-

```
a := b + c
z := a * a
w := a * a
u := 3
v := u + w
```

doing all the steps one more time, we get final minimal form:-

```
a := b + c
z := a * a
v := 3 + 2
```

Q.5: (a) call by value:-

O/p: p: 5, 3, 7

main: 2

(b) call by name:-

O/p: p: 5, 8, 8

main: 8

(c) call by reference:

O/p: p: 5, 8, 8

main: 8

(d) call by copy restore:

O/p: p: 5, 3, 7

main: 7

Nikeet Keshari

20174021

CS-A