2016/12/06 TUE (2:00)

I'm starting @ UNL later than I normally would, but I already did some coding @ home & also spent a little time w/ Jim & got distracted by Austin, so think I'm actually doing pretty ok overall.

> Question for later: if Austin were my son what would I do to cope?

85.3 average

78
93
+85
256
÷3

3|256
24
16
15
10

sum (sc.grade[i]...)÷3

(sum + sc.grade[i₁,₂,₃]) ÷3 counter

Counter++ till NULL @ "next"

p109 Create my own problem → one I can already do w/ array, but this time using heap memory & dynamic lengths...

- sort array [3, 7, 2, 4];
- avg, mode, total, etc of array #s
- input student grades or movie ratings, then do something w/ them (report highest / lowest score...)
- how many days practiced per day
  > report avg, best, worst, total min...

RE AUSTIN

1) find ways to only focus on the task at hand, not the giant overflowing pit of destruction @ is Austin's messed up life
   > dismiss the elements @ "aren't my problem", and actually follow through!
   > handle a problem one step at a time before attempting to anticipate everything & crashing

Austin's a crossword puzzle-level of difficulty - AND

don't internalize EVERYTHING

TLAP 4.2

abcdefg

input string: "abcdefg"
input position: 3
input length: 4
output: "cdef"

Substring ( ) {
(array String, int startPosition, int lengthOfChar)
return
}

pointer to new dynamically allocated string array.

original string unaffected by function

(if it were an array)

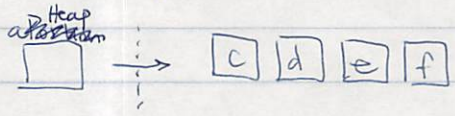
```
char a[7] = {'a', 'b', 'c', 'd', 'e', 'f', 'g'};
int stringPosition = 3;
int stringLength = 4;
for (int i=0; i<7; i++) {
write( a[stringPosi
}
for (int i = stringPosition-1; i<(stringLength+stringPosition-1); i++) {
 cout << a[i];
}
```

NE Zen Center
Omaha
~~tendth~~
> tendingtheox.com → Lincoln

2016/12/07 WED (noonish)

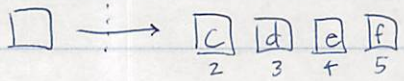
```
char arrayString[] = 'a,b,c,d,e,f,g'; (or similar syntax)
int StringPosition = 3;
int String length = 4;
```

return pointer to heap string array



a= a b c d e f g
0 1 2 3 4 5 6

new char[] aPortion = Heap




```
aPortion = aHeap;
~~aHeap~~ aHeap = 0;
return aPortion;  ?
```

~~void append (array~~

```
char ~~void~~ substring (char * array, int sPos, int sLgth) {
        ~~char * pArray = new char[slgth];~~
        char * pArray ~~output~~ = new char[sLgth];
        int count = 0;
        for (int i = sPos; i < (sLgth + sPos); i++){
            pArray ~~output~~ [count] = array[i];
            count ++;
        }
        char output[sLgth] = pArray ~~output~~;
        pArray ~~output~~ = 0;
        return output;
}
```

abcd xy

2017/01/15

ab bc cd xyz what have I already written?

abcd ⇒ axyd

abcd + xyz ⇒ abxyzcd

is there any more in between steps to more slowly increment from what I've successfully done and where I'm trying to go? ~~Jim suggested taking~~

- memory length: count up "b"'s and use @ count to tally final string length when new replacement characters are added in
- finding and replacing just ~~or~~ one-to-one letters, then adding the extra one in a second pass (b⇒x, then add y after x)

? - Run through array and target where "b" is → [0,1,00] →
~~not~~ take out "b" at 1 and replace w/ "xy"

quit thinking about if, else statements. What if I thought more in terms of how I wrote Jim's code. If if-else is too tricky, then find another insertion way. There are so many paths to a solution @ I should never limit myself to just one, especially if it's got me stuck. That's definitely something I've learned from past problem-solving attempts.

Even though I've got way more problems to solve w/ this assignment, maybe I should start w/ these mini-breakdowns and see where they take me and hopefully help me to build momentum.

=

create function @ takes #ofBs and assigns proper memory allocation for heap String. (abcd → xy ⇒ axycd
b → 4+1=5

2017/01/18

aLength = 4

~~repLength = 3~~ extraMemoryLength = @(3-1) ⇒ 2

numberOfBs = 1

newMemoryLength = 4 + (1×2) + 1 ⇒ 8

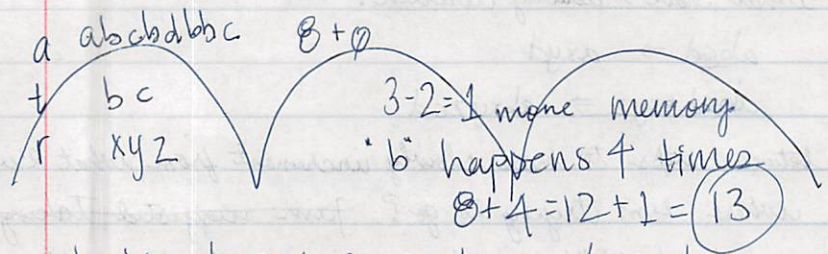
(actual memory needed → 7

abcd ⇒ axyzcd@

- right now, my functions only work if target is 1 character
also no ~~replace~~ one

~~to~~ noise-cancelling headphones.

~~12' × 13' 6"~~



• OK, so I've successfully created a quick-and-dirty heap memory computer & allocator. Now what?

→ take target & replace w/ replacement String.
Any micro-steps to get there??
→ make a 1-to-1 transfer, w/in my more complicated code wrapping? In short, drop in black box and make it fit & play nicely w/ other black boxes. Sounds good for a starting point!

```
if(String[i] != targ[0])
    outputString[i] = String[i];
else if (string[i] == targ[0])
    outputString[i] = ~~targ[0]~~ repl[0]
~~else~~
```

2017/01/~~19~~ 19 THUR

"axcd" → next step: find way to append "y" after "x". Read TLAP to try to glean some ideas. Also think about the mini-project Jim had me work on.

~~base[] = "abcd"~~
"axcd"
[find insert index → wherever "b" is]
int insertIndex = 1;

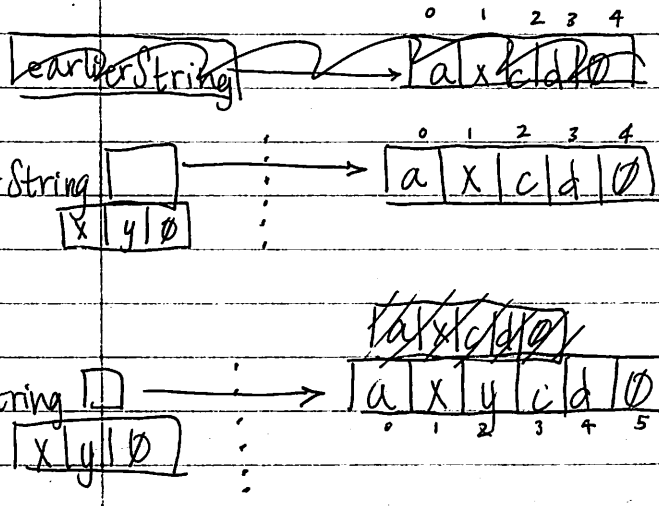
(possible method based on Jim's mini-project)

```
char* doInsert(char* string, char* targ, char* repl) {
    char* earlierString = outputFunc(string, targ, repl);   // "axcd"
    char* finalString = new char[newMemoryLength];
    int newMemoryLength = memoryLength(string, targ, repl);
    [Repeat For Loops in heap02SandboxD.cpp]
    Dereference earlierString;
    return finalString;
}
```

dereference/delete this

swap

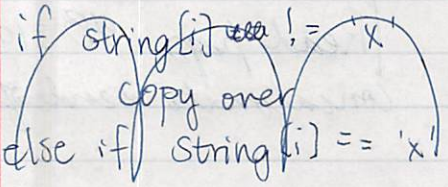
(2017/01/19)



[Read page 98-101]
Concatonate @ end of string

dereference earlierString;

2017/01/20 FRI



~~for (int i = 0; i <= indexInsert~~

```
char * doInsert(char * string, char * targ, Char * repl, int indexInsert) {
    int aLength = findStringLength(string); // base string length
    int stringInsertLength = findStringLength(repl); // replacement string length
    int newMemoryLength = memoryLength(string, targ, repl); // final string length

    char * earlierString = outputFunc(string, targ, repl); // "axcd" ← delete this var
    char * finalString = new char[newMemoryLength]; // heap var for final output string

    for (int i = 0; i < indexInsert; ; ++) {
        finalString[i] = earlierString[i];          // assign chars before insertIndex: beginning of baseString
    }
    for (int j = 0; j < (stringInsertLength - 1); j++) {      // -1 to account for first char
        finalString[j + indexInsert] = repl[j + 1];           // +1 already being included in earlierString for same reason as above
    }
    for (int k = indexInsert; k <= (aLength - indexInsert); k++) {
        finalString[k + indexInsert + stringInsertLength] = string[k + indexInsert];   // assign chars after insertIndex; remainder of base string
    }
    delete[] earlierString;
    return finalString;
}

int main() {
    :

char * finalOutputString = doInsert(a, target, replace, insertIndex);
```

minimimimimin!

2017/01/26 THUR

if targ = "bc", then we want "axydb"

"axcdx"

```
if string[i] != targ[0]
    (outputString[i] = string[i]) tallyArray[i] = 0;
else if string[i] == targ[0]
    (outputString[i] = repl[0]) tallyArray[i] = 1;
```

"abcdb" ⇒ "01010"
[this tallys all of the b's, not just first or last]

"01200"
a b c d b

~~if tallyArray[i]~~
~~if outputString[i] != repl[0]~~
~~finalOutputString[i] = output~~

```
if string[i] != targ[0]
    craneString[i] = ~~repl[0]~~ string[i];
else if string[i] == targ[0]
    checkFunction(string, targ, repl);
```

```
void checkFunction(string, targ, repl) {
    for (int j = 1; j < targLength; j++) {
        int K = 1; K < targLength; K++
        if string[K] == targ[j]
            ~~keep going~~ (outputVar = 1;)
        else if string[K] != targ[j]
            ~~string[i] = base~~ outputVar = 0;
            craneString[i] = string[i];
            break;
    ~~return output~~
```

~~delete~~, pass i as arg of checkFunc, but don't actually use it directly in iterating. I need it to be in its original place @ end of func if this doesn't match up!

How to keep going? What to do if all of the targs match string

→ If all of them match, then @ end of "check" loop, go to assignment function.

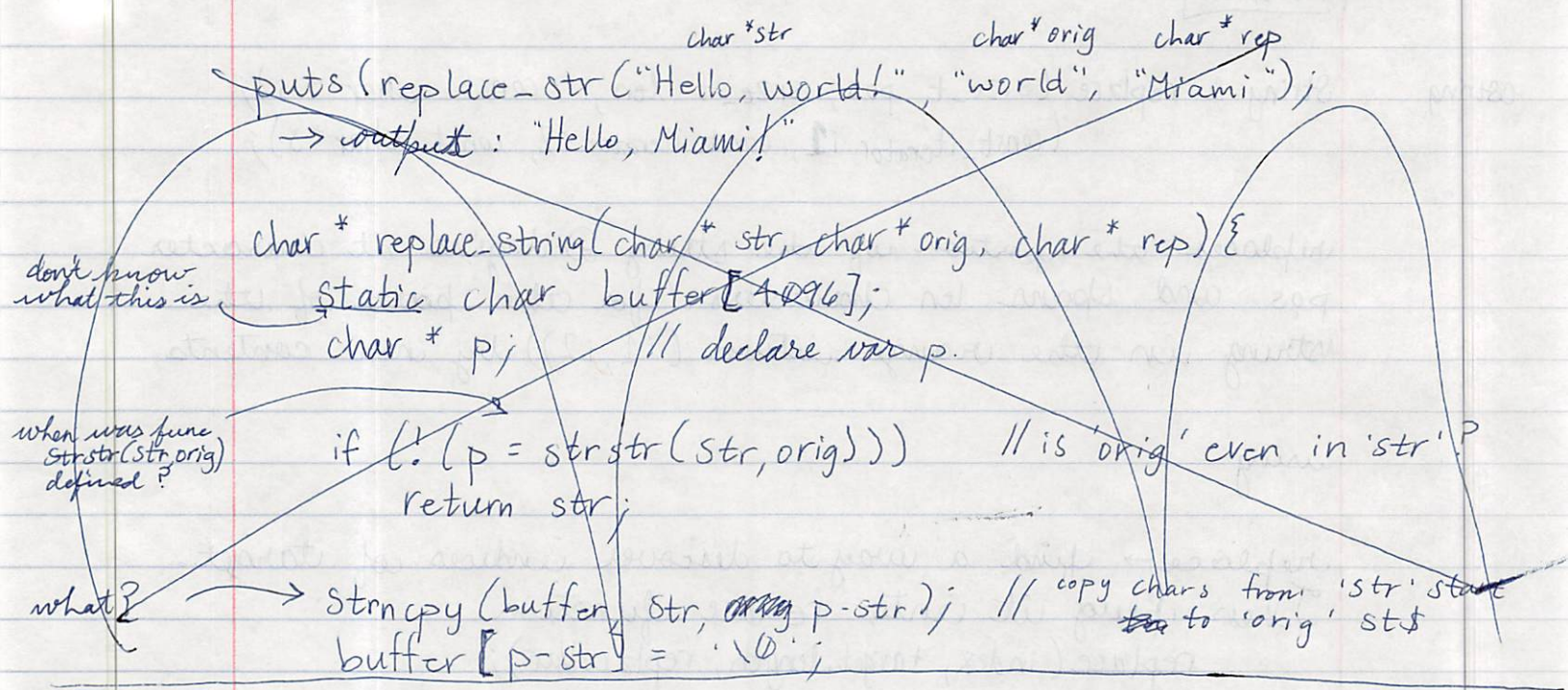
```
Maybe make 2 loop functions?
if outputVar == 1
    assignFunc(string, targ, repl, i);
```

[a b c d b \0]
0 1 2 3 4 5

i = 1

j = 2

2017/01/30 MON



FIND

include <string>

std::string::find

string → size_t find (const string& str, size_t pos = 0) const noexcept;

cstring → size_t find (const char* s, size_t pos = 0) const;

str ⇒ base string

pos ⇒ position of first character in str to be considered for search. (if greater than string length, the function never finds a match). Note, first char denoted by ~~char~~ value of 0 (not 1); a value of 0 means the whole string is searched (ie, start @ the beginning)

s ⇒ pointer to an array of characters. Using "cstring" version of function, [~~a null terminated sequence is expected~~] to know when to stop iterating ;; the length of the sequence to match is determined by the first occurrence of null character.

(size_t : ~~un~~signed integral type → the same as member type string::size_type)

Return value: the position of the first character of the first match. If no positions found, then returns a string::npos

REPLACE

cString

String & replace (size_t pos, size_t len, const char *s);
(const_iterator i1, const_iterator i2, const char*s);

replaces the portion of the string @ begins at character pos and spans len characters (for the part of the string in the range intwn (i1, i2)) by new contents

using

replace → find a way to discover indices of target. Then plug it into replace function
replace(index, target.length, replaceVar);
Call on to replace each time it needs to be performed!

122 = 18,5 Low normal

130 = 19.8

150 = 22.8

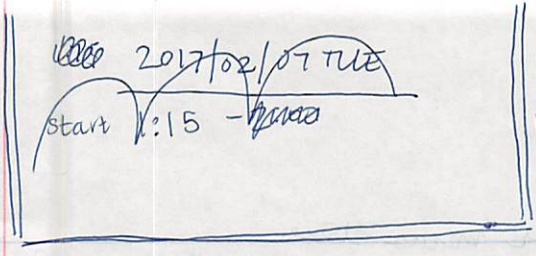
164 = 24.9 High normal

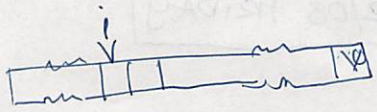
2017/02/03 FRIDAY


```
// returns 0 if false, any number means true
int matches_at(char* in_this_string, char * do_i_find_this_string,
    int at_this_index);

int main() {
    char base[] = "there once was a man from nantucket";
    char find[] = "a man from";
    if (matches_at(base, find, 15){
        cout << "found it where expected";
    } else {
        cout << "broken!";
    }
    return 0;
}

int where_at (in_this, do_i_find) {
 int which_index = -1;
 for (int i = 0; i < in_this.length(); i++) {
     if (matches_at(in_this, do_i_find)) {
         which_index = i;
     }
 }
 return which_index;
}
```





~~raw number~~

@ beginning of program, find where index of match is

initial scan func.

```
scan base
    if base[i] != target[0]
        outputVar[i] = base[i]
    else if
        base[i] = target[0]
{
if base[i] = target[0]
    matches_at_bool()
        → ~~if true, log index~~ log_matching_index();
        → ~~if false, move on.~~
    if matches_at_bool() == 1
        indexVar = i;
    ~~else~~ break;
}
```

~~[15, 20, 81];~~

scan base

```
for( i=0; i<baselength; i++) {
    if base[i] == targ[0] {
        run check to see if it's a full target
    }
}
if it's a full target
    log index & save it (indexVar = i;)
else if it isn't
    keep scanning.
```

```
// return:
// false = 0 ; true = any number
int matches_at(char * in_this_string, char * do_i_find_this_string,
int at_this_index) {
    int boo_in_the_zoo = 0;
    if
      in_this_string[i] != do_i_find[
  for( i=0 ; i < do_i_find.length() ; i++) {
      if(in_this_string[i + at_this_index] != do_i_find[i]){
          boo_in_the_zoo = 0;
          break;
      }else{
          boo_in_the_zoo = 1;
      }
  }
  return boo_in_the_zoo;
}
```

2017/02/07

```
func {
  for( )  {
indexVar = -1; → if(base(i) == target[0]){
          if (matches_at_bool() == 1) {
              indexVar = i; } or return i;
              break;
          }
    }
    return
} return indexVar; → or nothing return -1;
}
```

outputPlaceholder =

if no matches at all, @'s easy:
outputVar[i] = base[i]
end of story

if there have been matches, but this ending section has no more matches:

outputVar [

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44
"I say, hippos eat hippos on hippos for lunch Ø"

"dogs eat dogs on dogs for lunch."
"dogs"
"hippos"

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37
"I say, dogs eat dogs on dogs for lunch Ø"
"dogs"
"hippos"
0 1 2 3 4 5

0 1 2 3
dogs

Ø-5 Ø-5
output[i] = base[i] → "I say, "
basePlaceholder = i + target → 9

Ø-5 Ø 6 Ø-5
output[j + placeholder + matchingIndex] = replace[j] → "dogs" to "hippos"
outputPlaceholder = j + placeholder + matchingIndex → 11

output[i + basePlaceholder] = base[i + basePlaceholder] → " eat "
13-17 10-14
basePlaceholder = i + basePlaceholder + target → 18

Ø-5 11 6 Ø-5
output[j + outputPlaceholder + matchingIndex] = replace[j] → "dogs" to "hippos"
outputPlaceholder = j + outputPlaceholder + matchingIndex → 23

~~int outputIndexPlaceholder = 0;~~
~~for (int i = placeholder; i < matchingIndex~~

```
int placeholder = 0;
if ( matchingIndex >= 0) {
    for ( int i = placeholder; i < matchingIndex; i ++) {
        outputVar [i] = base[i];
    }
    for (int j = 0 ; j < replaceLength; j ++) {
        outputVar [j + placeholder + matchingIndex] = replaceVar[j];
        placeholder = j + placeholder + matchingIndex;
    }
else if ( matchingIndex < 0) {
```

(end of day 2017/02/07) I've successfully finished one pass through.
↳ Now figure out multiple loops of match, replace
Then figure out very end of phrase.

loop, then perform matchingIndex again, then loop again if true

```
loop do matchingIndex {
    if true {
        do transcription
    }
}
return outputVar
}
```

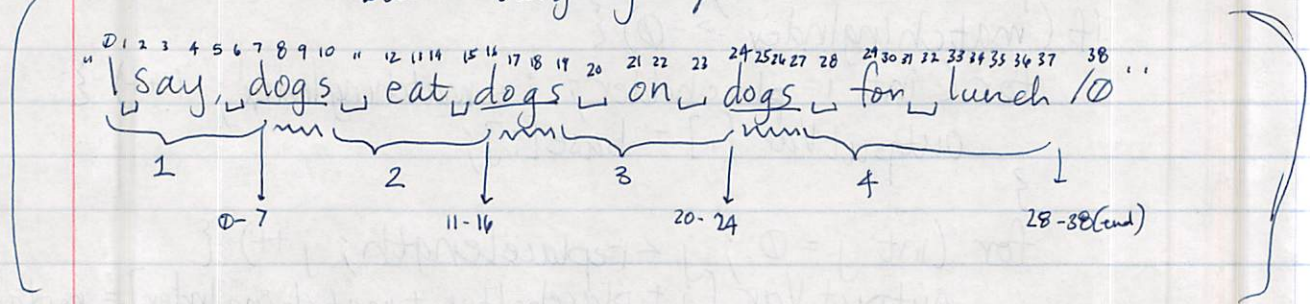
5 + 0 + 4 = 9

- no cellphone followup
- jump-start @ for him so he can take Uber?

fri 10 ($50 class subscription)
Community Action 210 "O"
taxes 1:30 (20 min)

log_matching_index(baseVar, targetVar, iteratorVar) {

passing over base, @ starting point
s/b/ ∅ @ beginning, then move forward based on base + targ jumps



matches_at_bool } pass basePlaceholder to each iterator value for i
log_matching_index

unless loop → unless matching index < ∅

bP	11	31
oP	13	35

first pass

75 i+∅ ; i+∅ (0-6)

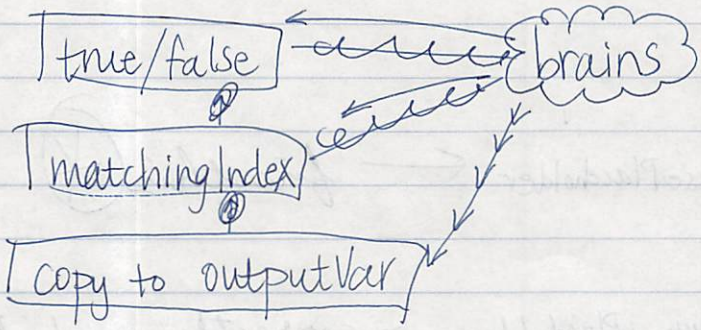
77 ~~7+0+4 → 10~~ 7+0+4 ⇒ ~~10~~ 11 → basePlaceholder

80 j+∅+7 ; j (0-5, 0-6)

82 ~~6+0+7~~ → (13) → outputPlaceholder

84 log_matching_index(baseVar, targetVar, basePlaceholder)
baseVar[11 starting @ "...gs cat do..."
returns matchingIndex (16)

2017/02/07 TUE



Iterate over outputVar.

- if matching index >= 0 → (do stuff)
  - check matching index multiple times until it's 1
  - when it's -1, copy over remainder of base and be done.

```
if(matchingIndex >= 0) {
    for(i = placeholder; i < matchingIndex; i++) {
        outputVar[i] = base[i];
    }
    for(int j=0; j < replaceLength; j++) {
        outputVar[j + placeholder + matchingIndex] = replaceVar[j];
        int placeholder = j + matchingIndex;
    }
}
```

placeholder = 0

this seems important! → outputIndex

but also need to remember outputVar.

also need to keep running checks if matchingIndex >= 0.

```
if(matchingIndex < 0) {
```

2 conditions:

- there's no match @ all
- there have been matches, but this (ending) section of cString has no more matches

outputVar[k] = [k + placeholder]

k = placeholder

outputVar[k] = base[k]

SECOND CYCLE

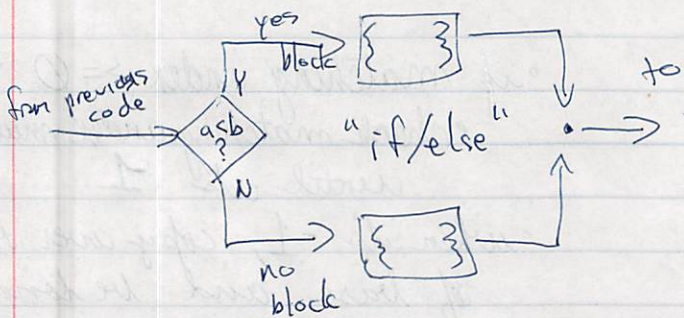
71 matchingIndex = 16

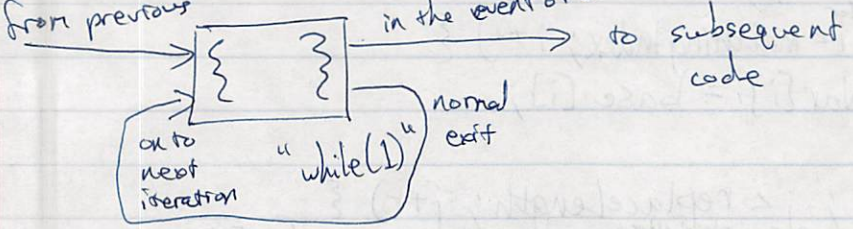
75 i+13 ; i+11 (i: 0-6)

77 7 + 11 + 4 → (22) basePlaceholder ← fix this (*)

80 j +13 +16 ; j (j: 0-5)

82 6 +13 +16 → (35) outputPlaceholder → correctly computed





set office hours
enforce them!

° Silence Phone, set auto-replies?

Some self-sadness, projected out @ Jim
→ I'm disappointing myself by not meeting my goals, but can't face that fact, so I needed to blame someone else.

2017/02/09 THUR

difference btwn basePlaceholder & outputPlaceholder

code: bP = i + bP + targetLength
code: oP = j + oP + matchingIndex

my math: 31 − 22 = 9 (difference)

matchingIndex = log_matching_index(baseVar, targetVar, basePlaceholder)

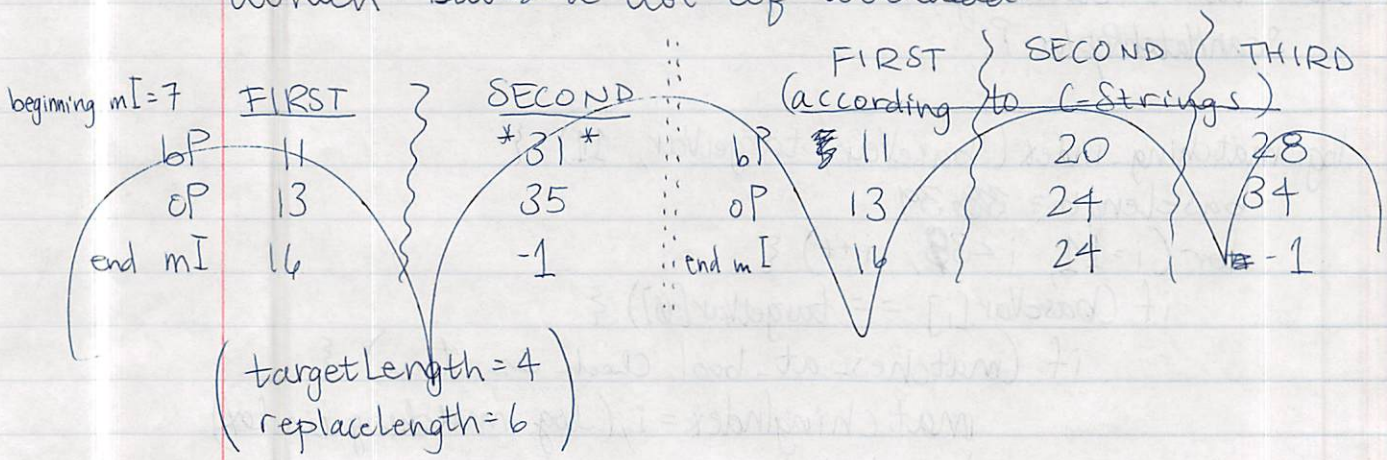
also as a curiosity, what if I enter raw numbers into ScanMatchReplace?

✓
```
log_matching_index( baseVar, targetVar, 11) {
    baseLength= 39
    for (i=11; i<39; i++) {
        if (baseVar[i] == targetVar[0]) {
            if (matches_at_bool check is true) {
                matchingIndex = i; (log matching index)
                break;
            }
        }
    }
    return -1 or index of match;
}
```

i (starts at 11 → in this case, it should pass 16)

✓
```
matches_at_bool (base, target, at_this_index) {
    targetLength = 4
    for (i=0; i<4; i++) {
        if (base [i+16] != targ [i]){
            boo_in_the_zoo = 0;   false
            break;
        }else{
            boo_in_the_zoo = 1;
        }
    } return
    return boo_in_the_zoo;
}
```

>> OK, my understanding of log_matching_index and of matches_at_bool matches what I see the computer doing. That suggests @ the problem is in ScanMatchReplace. I suspected @, but it's nice to have confirmation so @ I can limit my ~~scan~~ probing to @ func w/ confidence. I don't, for instance, need to modify the commented code in log_matching_index, which saves a lot of overhead.



results of updated code →

starting MI = 7
end of day 2017/02/09

start mI	FIRST		FIRST	SECOND	THIRD	
bP	11	Start mI	✓ 7	✓ 16	✓ 24	
oP	13	bP	✓ 11	✓ 20	✓ 28	yay!
end mI	16	oP	✓ 13	✓ 24	✓ 34	yay!
		end mI	✓ 16	✓ 24	✓ -1	

not right, despite all #s being exactly on!

output:
"I say, hippos eat dogs o on dogs for lunch" ?!?!!

(2017/02/09 THUR)

Ok, let's manually computi ScanMatchReplace a second time.

first round

what are my important variables?

oP = ∅ (0-6) i (∅) (7)

85 oP = j + oP + matchingIndex → 13

bP = ∅ (0-7) (∅) (4)

78 bP = i + bP + targetLength → 11

second round

oP = 13

~~85 oP = 7 6 + 13 + 13 = 39~~

85 oP = j (0-6) + oP (13) + matching Index (16) → ~~35~~ (sb 24)

(outputMatchingIndex + replaceLength - 1 + space btwn next mI)

7 5 6

bP

oldBaseMatchingIndex + targetLength - 1 + Space btwn next mI

7 3 6

(bP) 16 + 3 + gap btwn targets → 24

(mI) (tL-1) 5

(oP) mI + extraReplaceLength + rL-1 + gap btwn targets → 28

16 2 5 5

getting closer, but also getting confused about specifics...

(index = beginning of match character)
(@ an index ⓘ's logged)

```
while (matchingIndex >= 0){   //while there's a match to be had
    for (i = 0; i < matchingIndex; i++) {
        outputVar[i + oP] = baseVar[i + bP];
    }
```

iterate through length of mI — oV[i] = bV[i] transcribe non-target letters

oP: 0, ~~13~~, ~~24~~, 34 ␣ ␣ ␣ ␣

bP: 0, 11 ␣ ␣ ␣ ␣, 20 ␣ ␣ ␣ ␣, 28 ␣ ␣ ␣ ␣

iterate through length of replace chars (transcribe replace chars to oV)

```
for (j = 0; j < replaceLength; j++) {
    outputVar[j + mI + oP] = replaceVar[j];
}
```

reset mI for another round

```
matchingIndex = log_matching_index(base, target, newIndex);
```

of base — "d" "d" "d" (no more)

matchingIndex = 7, 16, 24, -1

corresponding output matchingIndex = 7, 18, 28, -1 (no more)

mI + (~~rL~~ rL - tL) / 2

bP = mI + tL ⇒ (11)
7 4

16 + 4 ⇒ (20)

24 + 4 ⇒ (28)

oP = mI + (~~rL - tL~~) + rL ⇒ (13)
(7 ~~+ 2~~)(rL+tL)×0 6

the first one doesn't need (rL - tL)

(16 + 2)^(rL+tL)+1) + 6 ⇒ (24)

(24 + 4)^((rL+tL)+2) + 6 ⇒ (34)

to keep track of:

- [x] · matching Index (dynamic; already logging)
- [x] · difference btwn rL & tL (static, not logging)
- [x] · number of passes through loop (dynamic, not logging)
- [ ] ? · change oP formula
- [ ] ! · change bP formula

(2017/02/09 Thur)

63 `for ( j=0; j<replaceLength; j++) {`
`outputVar[j+outputPlaceholder+matchingIndex] = replaceVar[j];`
-02- `outputVar[j + oP + mI + (charDiffLgth * passesThroughLoop)] = rV[j];`

outputVar replaceVar

first pass → ((0-5) + 0 + 7) → (7→12) = (0→5)
second pass (0-5) + 13 + 16 → (29→34) way off! = (0→5)

↳ needs to be (18-23)

formula should be → [j + mI + (charDiffLgth * passesThroughLoop)]

outputVar replaceVar

first pass (0-5) + 0 + 7 + (2×0) → (7→12) = (0→5)
second pass

if that's the formula:

63 `for ( j=0; j<replaceLength; j++) {`
`outputVar[j+matchingIndex+(charDiffLgth*passesThroughLoop)] = replaceVar[j];`

oV rV

first pass oV[(0-5) + 7 + (2×0)] → (7→12) = [0→5]
second pass oV[(0-5) + 16 + (2×1)] → (18→23) = [0→5]
third pass oV[(0-5) + 24 + (2×2)] → (28→33) = [0→5]

(3:00PM)

2017/02/13 MONDAY

- log-matching_index has decided ⊕ there is no further match
- I want to transcribe the remainder of base over to outputVar

```
for (int i = 0; i < baseLength; i++) {
    outputVar[i + outputPlaceholder] = baseVar[i + basePlaceholder];
}
```

```
while (keepGoing == true)
    if (mI >= 0) {
    }
    else {
    }
    keepGoing = false;
    return outputVar;
}
```

```
while (keepGoing) {
  if (matchingIndex >= 0) {
    for (i=0; i<mI; i++) {
      oV[i+oP] = bV[i+bP];
    }
    bP++;
  }else {
    for (k=0; k<baseLength; k++) {
      oV[k+oP] = bV[k+bP];
    }
    keepGoing = 0;
    break;
  }
```

```
while (keepGoing) {
  if (matchingIndex >= 0) {
    for (i=0; i<matchingIndex; i++) {
      outputVar[i + outputPlaceholder] = baseVar[i + basePlaceholder];
    }
    bP++
  }else {
    for (k=0; k<=baseLength; k++) {
      outputVar[k+oP] = baseVar[k+bP];
    }
    if (k < baseLength) {
      outputVar[k+oP] = baseVar[k+bP];
    }else {
      break;
    }
  keepGoing=0;
  break;
}
```

heap →