```c
typedef struct Str {
        int bytes;
        char* data;
} Str;
```

Let's write split in our context with Str. Let's say delimeter is a single char. Should return array of Str

error: not allowed Str[]

what return type

Str* split(Str s, char delim)

A: List (stay tuned)

B: ~~char**~~

C: ~~Str~~

D: ~~Str[]~~

E: Str**

F: ~~char*[]~~

G: None of the above

```c
Str s = new_Str("miles joe nick");
Str* lst = split(s, ' ');
```

Str {"miles", "joe", "nick"}

what variable decl?

arrows are for assigning and dots are for accessing right?

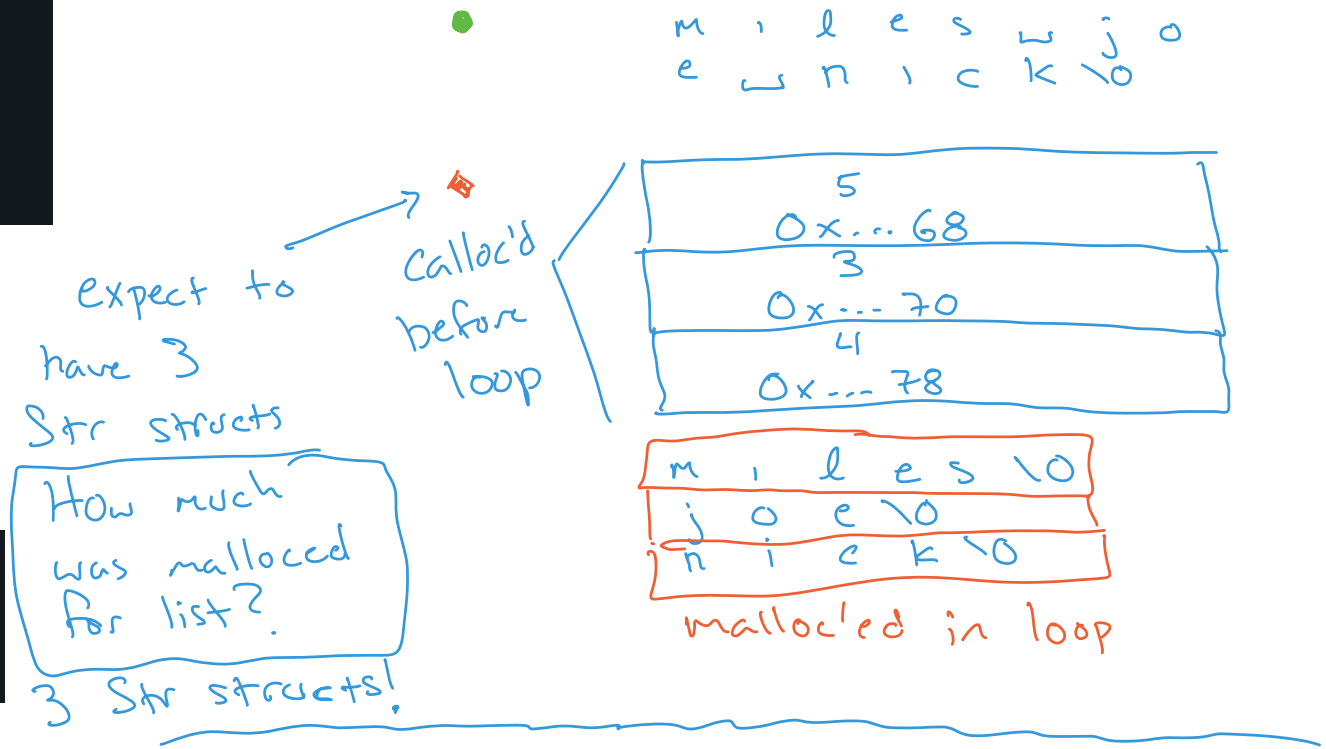
No!

value.x expect value is a struct type with x field

value->x expect value is T* and T is a struct type

```c
typedef struct Str {
        int bytes;      // Does not count the \0
        char* data;
} Str;

Str str(char* init) {
        int len = strlen(init);
        char* data = malloc(len + 1);
        strcpy(data, init);
        Str s = { len, data };
        return s;
}
```

```c
Str s = str("miles joe nick");
Str* list = split(s, ' ');
printf("%s %d\n", list[0].data, list[0].bytes);
printf("%s %d\n", list[1].data, list[1].bytes);
printf("%s %d\n", list[2].data, list[2].bytes);
```



but how does Str* type know when the array ends. char* has the null terminator, what about here? Or does malloc do that for us?

Str* does not tell us when the array ends

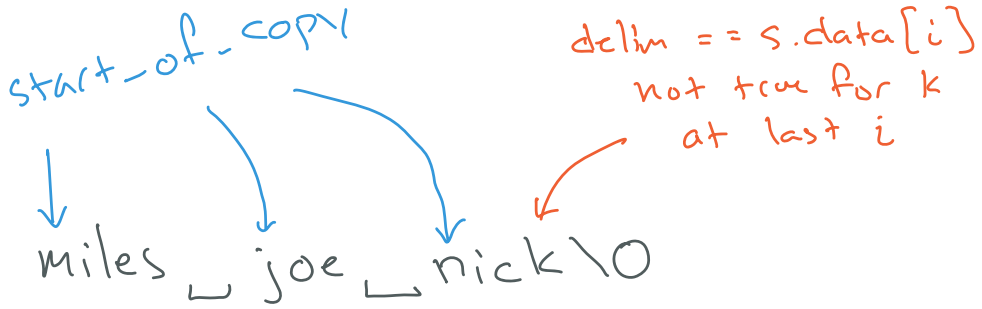
```
Str* split(Str s, char delim) {
        // allocate the array to return
        // put string data into the array
        // find instances of delim within the Str s

        // What size to malloc?
        // Number of delimiters plus 1 times size of a Str
        // A Str always has the same size
        int count = 0;
        for(int i = 0; i < s.bytes; i += 1) {
                if(s.data[i] == delim) { count += 1; }
        }
        Str* toReturn = calloc(sizeof(Str), count + 1);
        printf("Allocated %ld bytes\n", sizeof(Str) * (count + 1));
        // Does basically what malloc does
        // (multiplication inside calloc)
        // BUT calloc sets all the contents to 0
        // Str* toReturn = malloc(sizeof(Str) * count);

        return toReturn;
}
```

```c
Str* split(Str s, char delim) {
  // put string data into the array
  // find instances of delim within the Str s
  int count = 0;
  for(int i = 0; i < s.bytes; i += 1) {
    if(s.data[i] == delim) { count += 1; }
  }
  Str* toReturn = calloc(sizeof(Str), count + 1);

  int start_of_copy = 0;
  int split_index = 0;
  for(int i = 0; i < s.bytes; i += 1) {
    if(s.data[i] == delim) {
      int size = (i - start_of_copy) + 1;
      char* new_str = malloc(size);
      strncpy(new_str, s.data + start_of_copy, size - 1);
      new_str[size - 1] = 0;
      start_of_copy = i + 1;
      Str new_s = { size - 1, new_str };
      toReturn[split_index] = new_s;
      split_index += 1;
    }
  }

  return toReturn;
}


int main() {


  Str s = str("miles joe nick");
  Str* list = split(s, ' ');
  printf("%s %d\n", list[0].data, list[0].bytes);
  printf("%s %d\n", list[1].data, list[1].bytes);
  printf("%s %d\n", list[2].data, list[2].bytes);

```

```
$ gcc -g str.c -o str
$ ./str
miles 5
joe 3
(null) 0
```

there is no delim after nick, so the if statement is never being met for "nick"

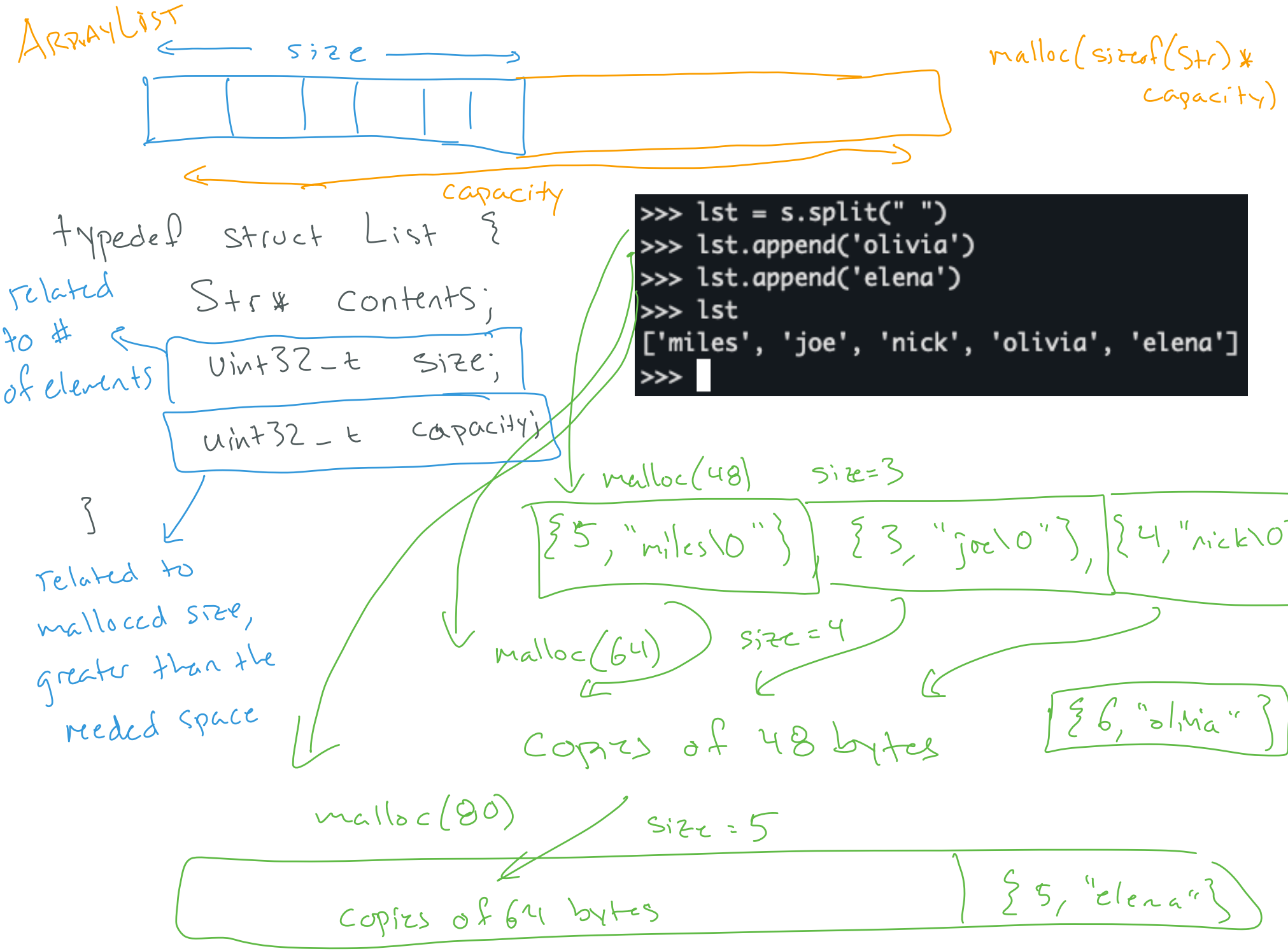
the last part of the string does not have a delimiter so the if condition cannot detect that last part

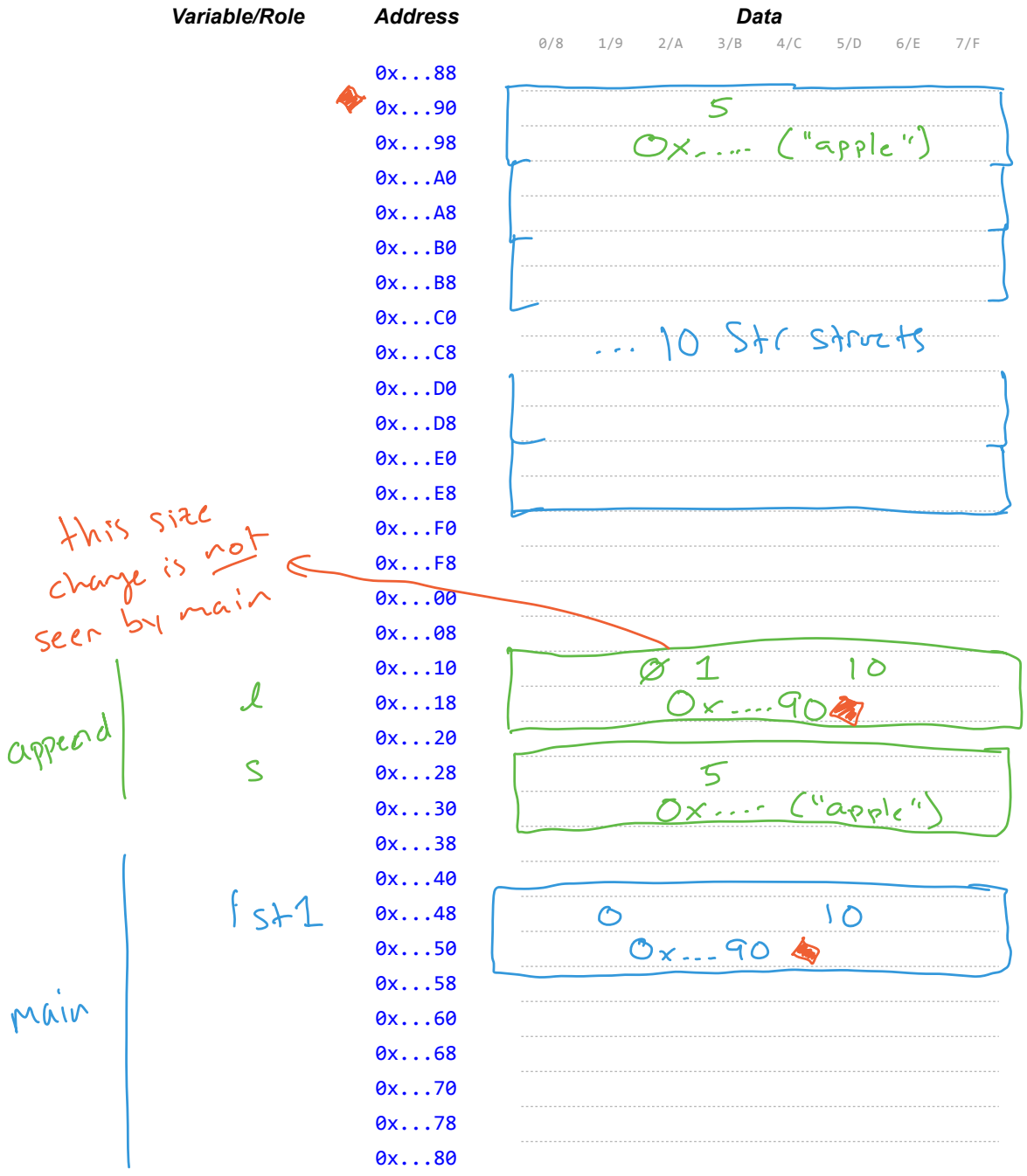


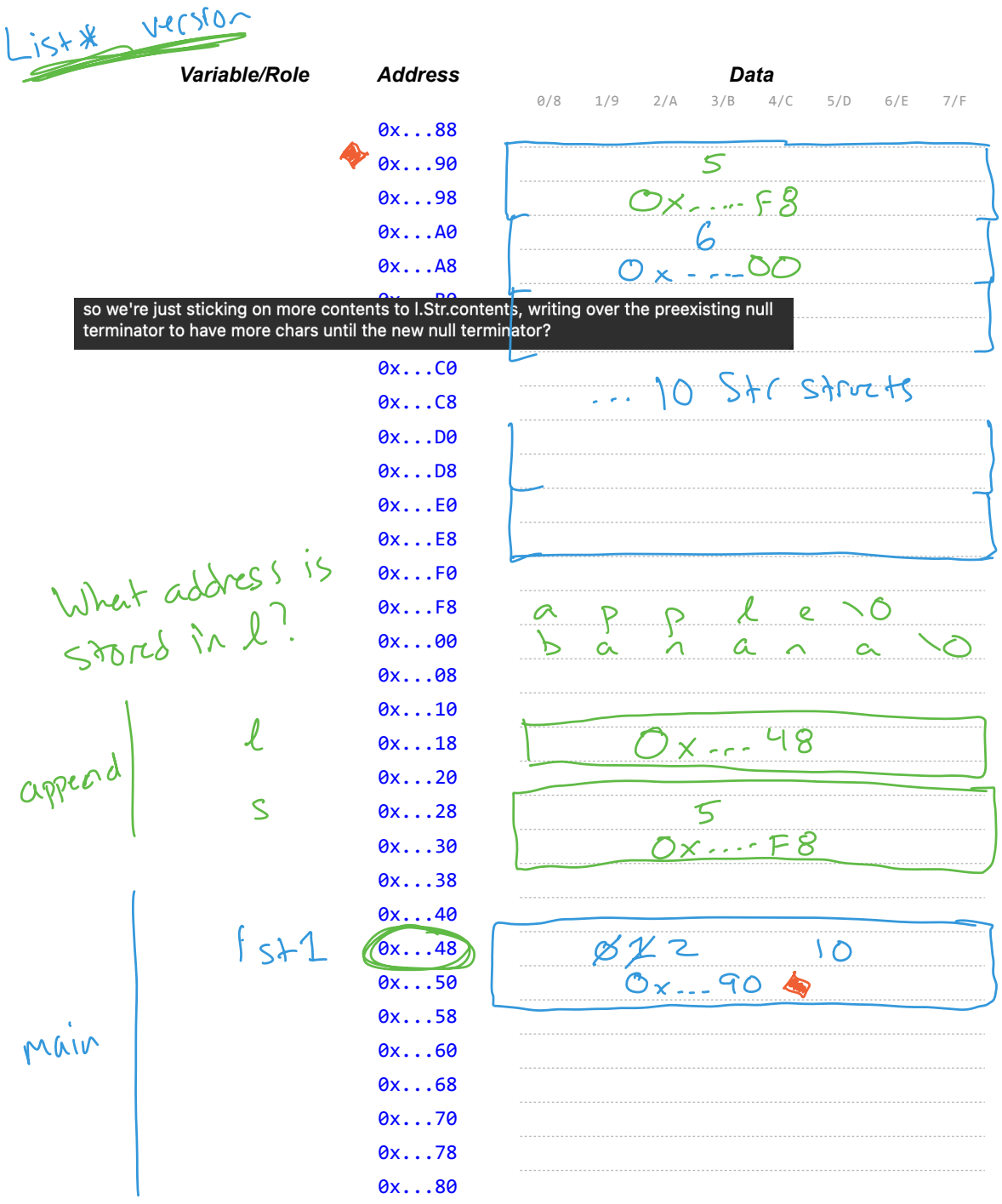
```
44
45 Str* split(Str s, char delim) {
46   // put string data into the array
47   // find instances of delim within the Str s
48   int count = 0;
49   for(int i = 0; i < s.bytes; i += 1) {
50     if(s.data[i] == delim) { count += 1; }
51   }
52   Str* toReturn = calloc(sizeof(Str), count + 1);
53
54   int start_of_copy = 0;
55   int split_index = 0;
56   for(int i = 0; i <= s.bytes; i += 1) {
57     if(s.data[i] == delim || s.data[i] == 0) {
58       int size = (i - start_of_copy) + 1;
59       char* new_str = malloc(size);
60       strncpy(new_str, s.data + start_of_copy, size - 1);
61       new_str[size - 1] = 0;
62       start_of_copy = i + 1;
63       Str new_s = { size - 1, new_str };
64       toReturn[split_index] = new_s;
65       split_index += 1;
66     }
67   }
68
69   return toReturn;
70 }
71
72
73 int main() {
74   Str s = str("miles joe nick");
75   Str* list = split(s, ' ');
76   printf("%s %d\n", list[0].data, list[0].bytes);
77   printf("%s %d\n", list[1].data, list[1].bytes);
78   printf("%s %d\n", list[2].data, list[2].bytes);
79
80   Str oneword = str("cse29");
81   Str* list1 = split(oneword, ' ');
82   printf("%s %d\n", list1[0].data, list1[0].bytes);
83
84   Str multispace = str("apple   banana");
85   Str* list2 = split(multispace, ' ');
86   printf("%s %d\n", list2[0].data, list2[0].bytes);
87   printf("%s %d\n", list2[1].data, list2[1].bytes);
88   printf("%s %d\n", list2[2].data, list2[2].bytes);
89   printf("%s %d\n", list2[3].data, list2[3].bytes);
```

Why no free(s.data) here?

```
$ gcc -g str.c -o str
$ ./str
miles 5
joe 3
nick 4
cse29 5
$ gcc -g str.c -o str
$ ./str
miles 5
joe 3
nick 4
cse29 5
apple 5
 0
 0
banana 6
$
```


ARRAYLIST
size
capacity
malloc(sizeof(Str) * capacity)
typedef struct List {
Str* contents;
uint32_t size;
uint32_t capacity;
}
related to # of elements
related to malloced size, greater than the needed space
>>> lst = s.split(" ")
>>> lst.append('olivia')
>>> lst.append('elena')
>>> lst
['miles', 'joe', 'nick', 'olivia', 'elena']
>>>
malloc(48)
size=3
{5, "miles\0"}
{3, "joe\0"}
{4, "nick\0}
malloc(64)
size=4
copies of 48 bytes
{6, "olivia"}
malloc(80)
size=5
copies of 64 bytes
{5, "elena"}


Variable/Role
Address
Data
0/8 1/9 2/A 3/B 4/C 5/D 6/E 7/F
0x...88
0x...90
0x...98
0x...A0
0x...A8
0x...B0
0x...B8
0x...C0
0x...C8
0x...D0
0x...D8
0x...E0
0x...E8
0x...F0
0x...F8
0x...00
0x...08
0x...10
0x...18
0x...20
0x...28
0x...30
0x...38
0x...40
0x...48
0x...50
0x...58
0x...60
0x...68
0x...70
0x...78
0x...80
5
0x.... ("apple")
... 10 Str structs
this size change is not seen by main
append
l
s
fst1
main
0 1 10
0x....90
5
0x.... ("apple")
0 10
0x...90



so we're just sticking on more contents to l.Str.contents, writing over the preexisting null terminator to have more chars until the new null terminator?

For preventing memory leaks, should we handle it in main method or in the code method better?