CS107, Lecture 10 Extra Practice
Arrays and Pointers, Take II

Reading: K&R (5.2-5.5) or Essential C section 6

Ed Discussion: https://edstem.org/us/courses/46162/discussion/3643757

1. char* vs char[] exercises

Suppose we use a variable `str` as follows:

```
    // initialize as below
A   str = str + 1;
B   str[1] = 'u';
C   printf("%s", str)
```

For each of the following initializations:

- Will there be a compile error/segfault?
- If no errors, what is printed?

1.
```
char str[7];
strcpy(str, "Hello1");
```

2.
```
char *str = "Hello2";
```

3.
```
char arr[7];
strcpy(arr, "Hello3");
char *str = arr;
```

4.
```
char *ptr = "Hello4";
char *str = ptr;
```

1. char* vs char[] exercises

Suppose we use a variable `str` as follows:

```
   // initialize as below
A  str = str + 1;
B  str[1] = 'u';
C  printf("%s", str)
```

For each of the following initializations:

- Will there be a compile error/segfault?
- If no errors, what is printed?

1.
```
char str[7];
strcpy(str, "Hello1");
```

Line A: Compile error (cannot reassign array)

2.
```
char *str = "Hello2";
```

Line B: Segmentation fault (string literal)

3.
```
char arr[7];
strcpy(arr, "Hello3");
char *str = arr;
```

Prints eulo3

4.
```
char *ptr = "Hello4";
char *str = ptr;
```

Line B: Segmentation fault (string literal)

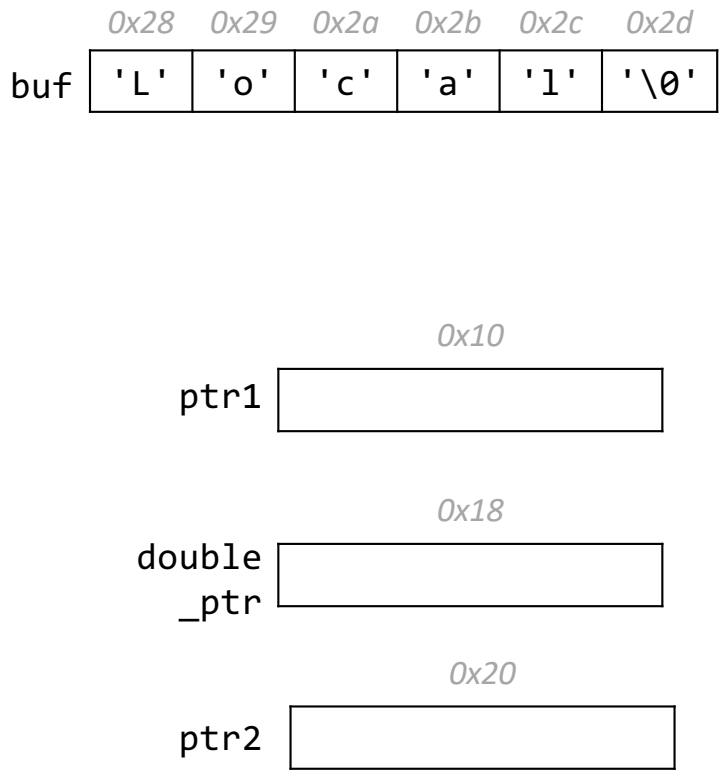
2. Bonus: Tricky addresses

```
1  void tricky_addresses() {
2    char buf[] = "Local";
3    char *ptr1 = buf;
4    char **double_ptr = &ptr1;
5    printf("ptr1's value:     %p\n", ptr1);
6    printf("ptr1’s deref    : %c\n", *ptr1);
7    printf("       address:   %p\n", &ptr1);
8    printf("double_ptr value: %p\n", double_ptr);
9    printf("buf's address:    %p\n", &buf);

10   char *ptr2 = &buf;
11   printf("ptr2's value:     %s\n", ptr2);
12 }
```

What is stored in each variable?

2. Bonus: Tricky addresses

```
1  void tricky_addresses() {
2    char buf[] = "Local";
3    char *ptr1 = buf;
4    char **double_ptr = &ptr1;
5    printf("ptr1's value:     %p\n", ptr1);
6    printf("ptr1’s deref    : %c\n", *ptr1);
7    printf("       address:   %p\n", &ptr1);
8    printf("double_ptr value: %p\n", double_ptr);
9    printf("buf's address:    %p\n", &buf);

10   char *ptr2 = &buf;
11   printf("ptr2's value:     %s\n", ptr2);
12 }
```



While Line 10 raises a compiler warning, functionally it will still work—because pointers are **addresses**.

Translating C into English

* If **declaration**: "pointer"
ex: `int *` is "pointer to an int"
If **operation**: "dereference/the value at address"
ex: `*num` is "the value at address num"

& "address of"

`<ptr name>` address

`<arr name>` address (except sizeof)

```
int arr[] = {3, 4, -1, 2};
// initializes stack array
// with 4 ints
```

1. `int *ptr0 = arr;`
2. `int *elt0 = *arr;`
3. `int elt = *(arr + 3);`
4. `int **ptr1 = &ptr0;`

Type check with a diagram!

Translating C into English

* If **declaration**: "pointer"
ex: `int *` is "pointer to an int"
If **operation**: "dereference/the value at address"
ex: `*num` is "the value at address num"

& "address of"

`<ptr name>` address

`<arr name>` address (except sizeof)

```
int arr[] = {3, 4, -1, 2};
```

1. `int *ptr0 = arr;`
2. `int *elt0 = *arr;`
3. `int elt = *(arr + 3);`
4. `int **ptr1 = &ptr0;`

```
// initializes stack array
// with 4 ints
```

Address arr

Value at address arr

The value at address <3 ints after address arr>

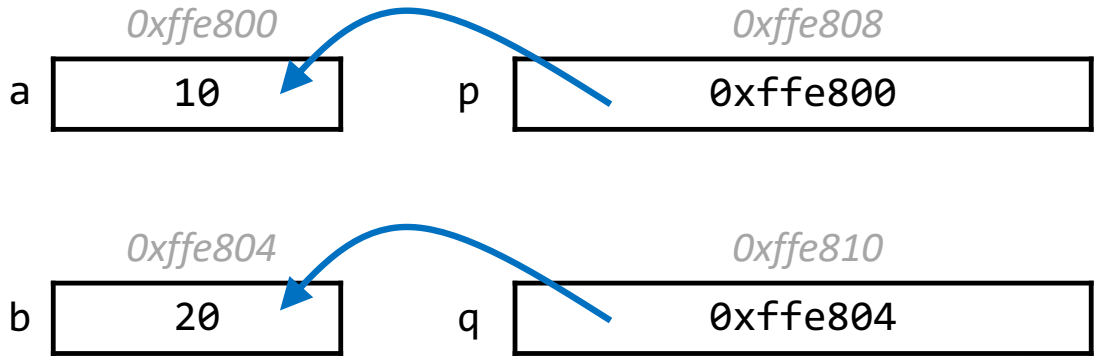
address of ptr

Type check with a diagram!

Pen and paper: A * Wars Story

```
1 void binky() {
2     int a = 10;
3     int b = 20;
4     int *p = &a;
5     int *q = &b;
6
7     *p = *q;
8     p = q;
9 }
```

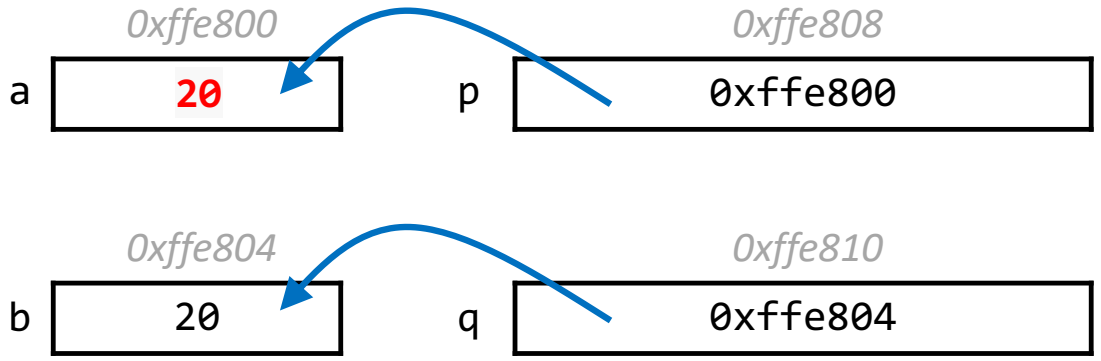
- Lines 2-5: Draw a diagram.
- Line 7: Update your diagram.
- Line 8: Update your diagram.



Pen and paper: A * Wars Story

```
1 void binky() {
2     int a = 10;
3     int b = 20;
4     int *p = &a;
5     int *q = &b;
6
7     *p = *q;
8     p = q;
9 }
```

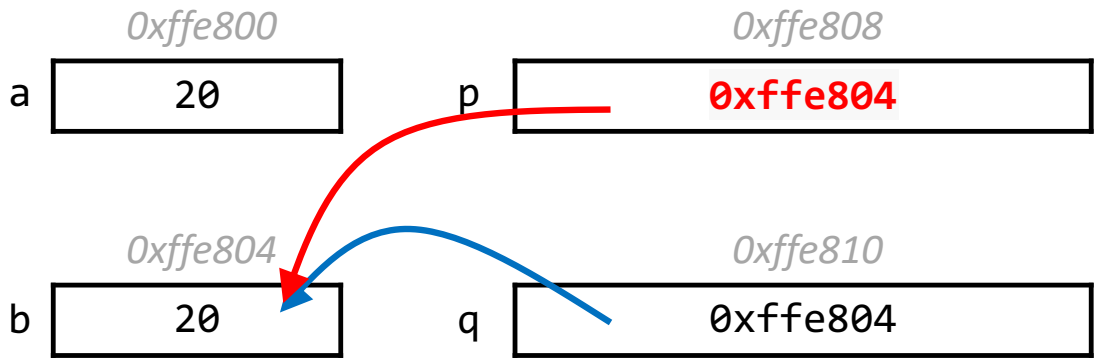
- Lines 2-5: Draw a diagram.
- Line 7: Update your diagram.
- Line 8: Update your diagram.



Pen and paper: A * Wars Story

```
1 void binky() {
2     int a = 10;
3     int b = 20;
4     int *p = &a;
5     int *q = &b;
6
7     *p = *q;
8     p = q;
9 }
```

- Lines 2-5: Draw a diagram.
- Line 7: Update your diagram.
- Line 8: Update your diagram.



* Wars: Episode I (of 2)

In variable **declaration**, * creates a **pointer**.

```
char ch = 'r';
```

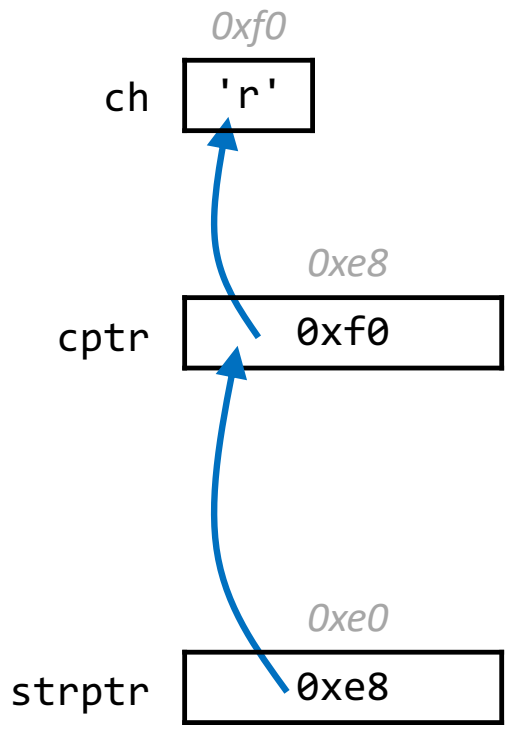
ch stores a char

```
char *cptr = &ch;
```

cptr stores an address of a char
(**points to** a char)

```
char **strptr = &cptr;
```

strptr stores an address of a char *
(**points to** a char *)

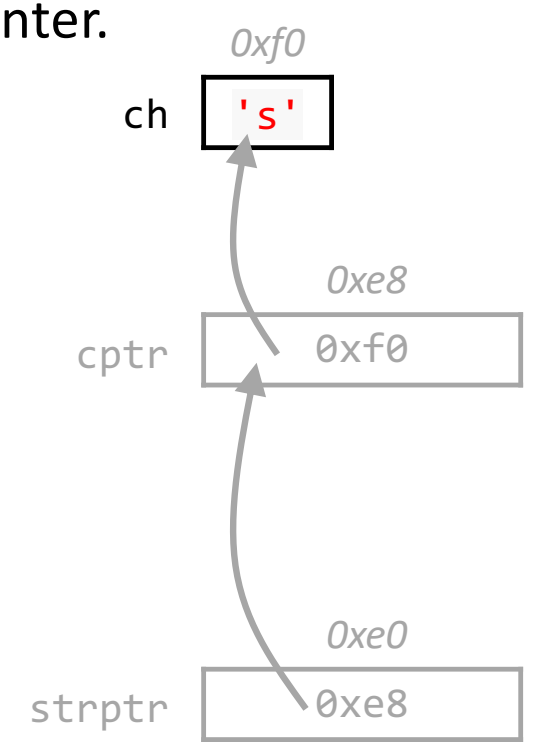


* Wars: Episode II (of 2)

In **reading values from/storing values**, * **dereferences** a pointer.

```
char ch = 'r';
ch = ch + 1;

char *cptr = &ch;


char **strptr = &cptr;
```

Increment value stored in ch



* Wars: Episode II (of 2)

In **reading values from/storing values**, * **dereferences** a pointer.

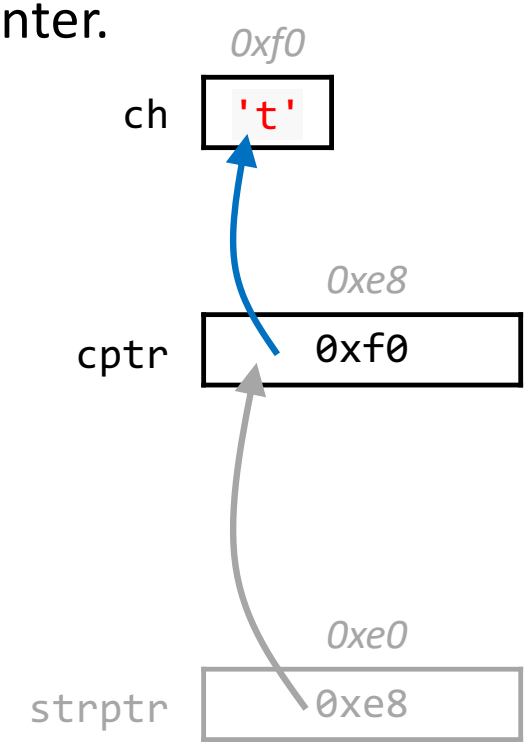
```
char ch = 'r';
ch = ch + 1;
```

Increment value stored in ch

```
char *cptr = &ch;
*cptr = *cptr + 1;
```

Increment value stored at memory address in cptr (increment char **pointed to**)

```
char **strptr = &cptr;
```



* Wars: Episode II (of 2)

In **reading values from/storing values**, * **dereferences** a pointer.

```
char ch = 'r';
ch = ch + 1;
```

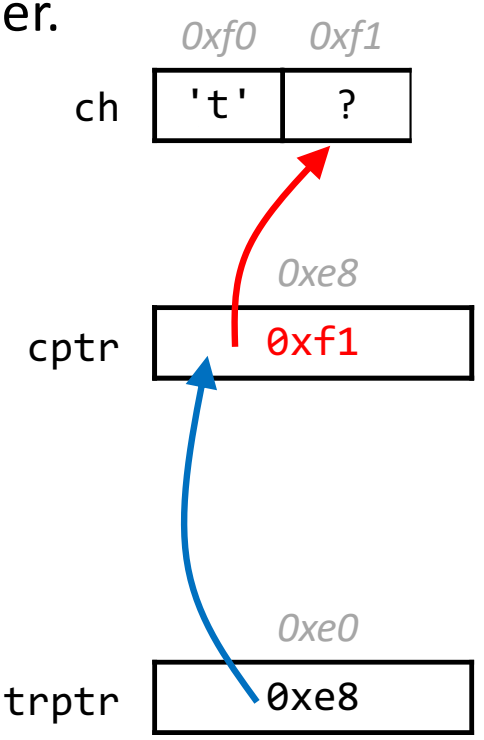
Increment value stored in ch

```
char *cptr = &ch;
*cptr = *cptr + 1;
```

Increment value stored at memory address in cptr (increment char **pointed to**)

```
char **strptr = &cptr;
*strptr = *strptr + 1;
```

Increment value stored at memory address in cptr (increment address **pointed to**)



Exercise: Implementation

The below function sums up the string lengths of the num strings in `strs`.

- Try both 1. array [ ] syntax and 2. pointer arithmetic!

```
1 size_t get_total_strlen(char *strs[], size_t num) {
2     size_t total_length = 0;
3     for (int i = 0; i < num; i++) {
4         // fill this in
5     }
6     return total_length;
7 }
```

Exercise: Implementation

The below function sums up the string lengths of the num strings in `strs`.

- Try both 1. array [ ] syntax and 2. pointer arithmetic!

```
1 size_t get_total_strlen(char *strs[], size_t num) {
2     size_t total_length = 0;
3     for (int i = 0; i < num; i++) {
4         // TODO: fill this in two ways
5     }
6     return total_length;
7 }
```

Equivalent:

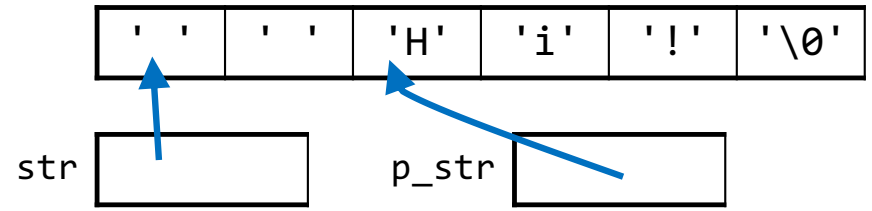
1. `total_length += strlen(strs[i]);`
2. `total_length += strlen(*(strs + i));`

Skip spaces

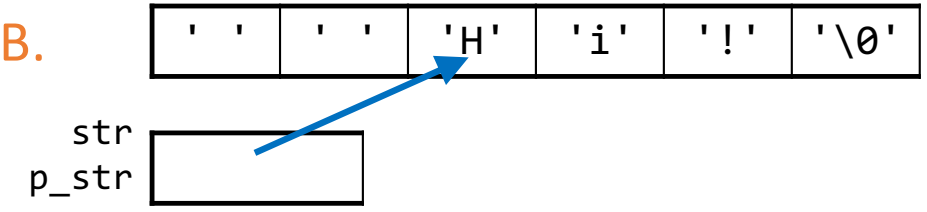
```
1  void skip_spaces(char **p_str) {
2    int num = strspn(*p_str, " ");
3    *p_str = *p_str + num;
4  }
5  int main(int argc, char *argv[]){
6    char *str = "  Hi!";
7    skip_spaces(&str);
8    printf("%s", str); // "Hi!"
9    return 0;
10 }
```

What diagram most accurately depicts program state at Line 4 (before `skip_spaces` returns to `main`)?

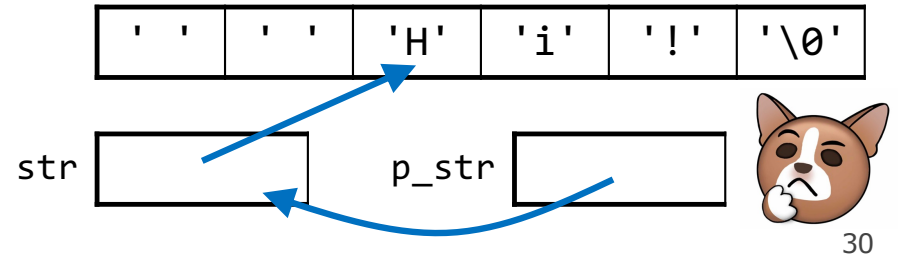
A.



B.



C.

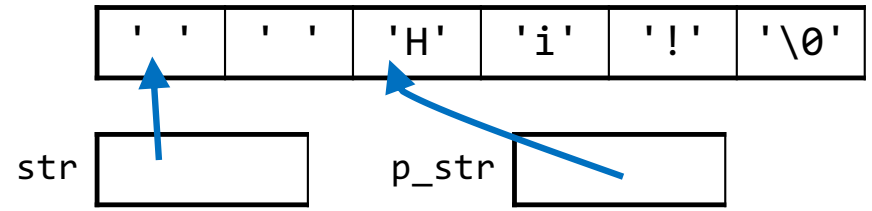


Skip spaces

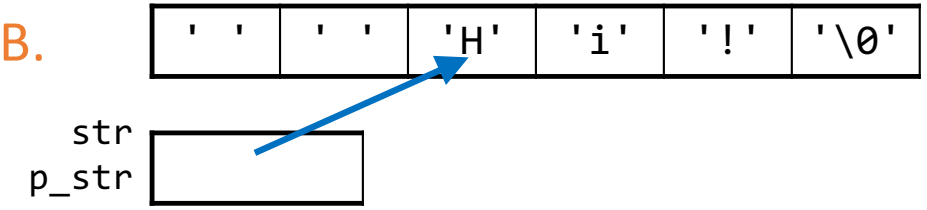
```
1  void skip_spaces(char **p_str) {
2    int num = strspn(*p_str, " ");
3    *p_str = *p_str + num;
4  }
5  int main(int argc, char *argv[]){
6    char *str = "  Hi!";
7    skip_spaces(&str);
8    printf("%s", str); // "Hi!"
9    return 0;
10 }
```

What diagram most accurately depicts program state at Line 4 (before `skip_spaces` returns to `main`)?

A.



B.



C.