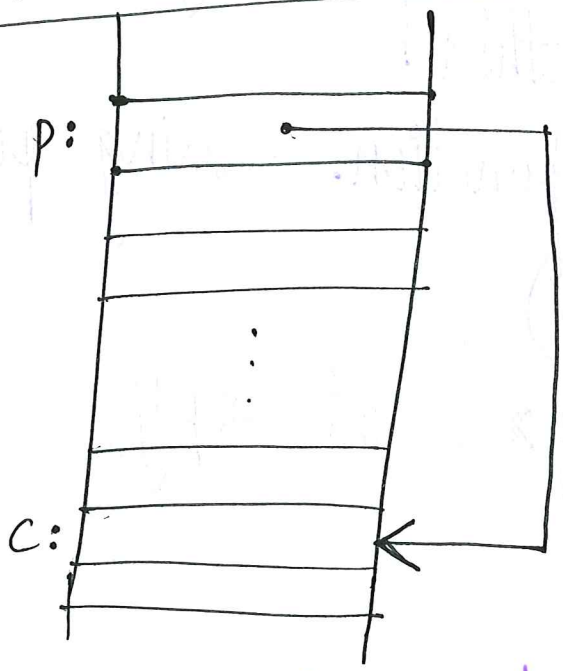
Pointers! (1)

A pointer is a variable that contains the address of a variable.

(1) Pointers and Addresses




```
p = &c;
```

```
*p = *(&c) = c
```

show live code!

& - address of operator.

```
p = &c;
```

⇒ applies only to objects in memory.
eg. variables and array elements.

⇒ cannot be applied to .
- expressions
- constants
- register variables.

* - indirection or ~~defer~~ dereferencing operator.

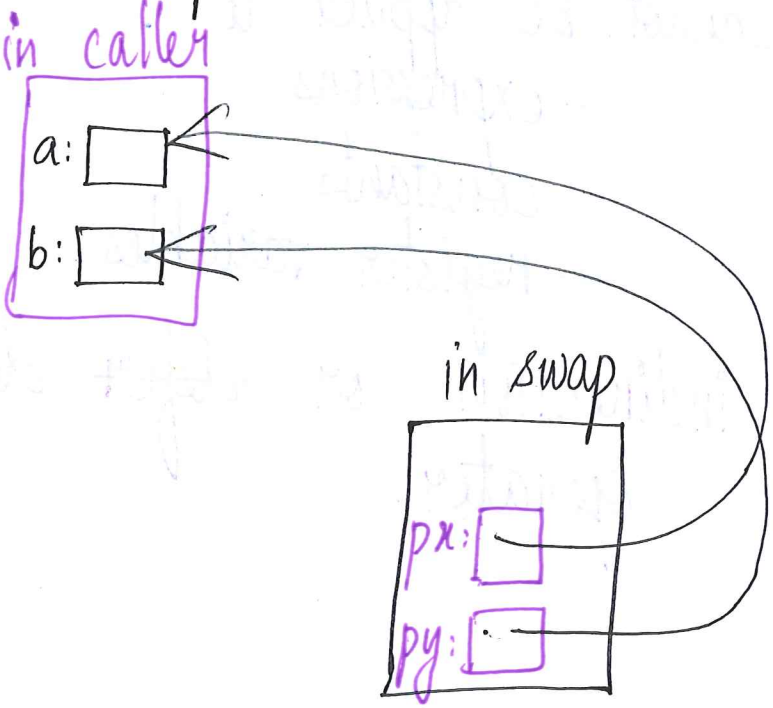
(2.) Pointers and Function Arguments

* C passes arguments to functions by value.
* WRONG swap method!
* CORRECT swap function. - using pointers!

swap(&a, &b)

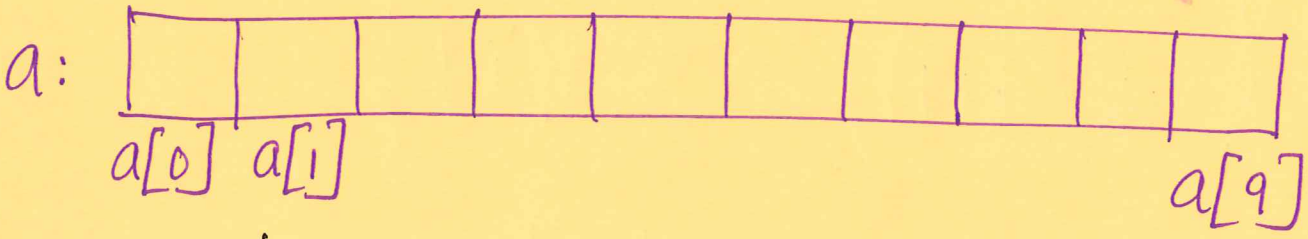
```
void swap (int *px, int *py)
{   int temp;

    temp = *px;
    *px = *py;
    *py = temp;
}
```

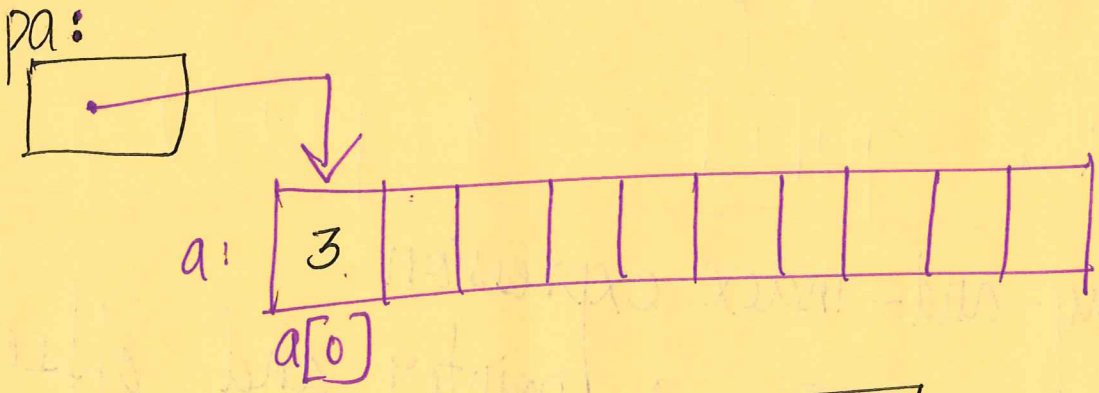


(3.) Pointers and Arrays

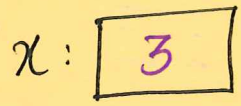
```
int a[10];
```

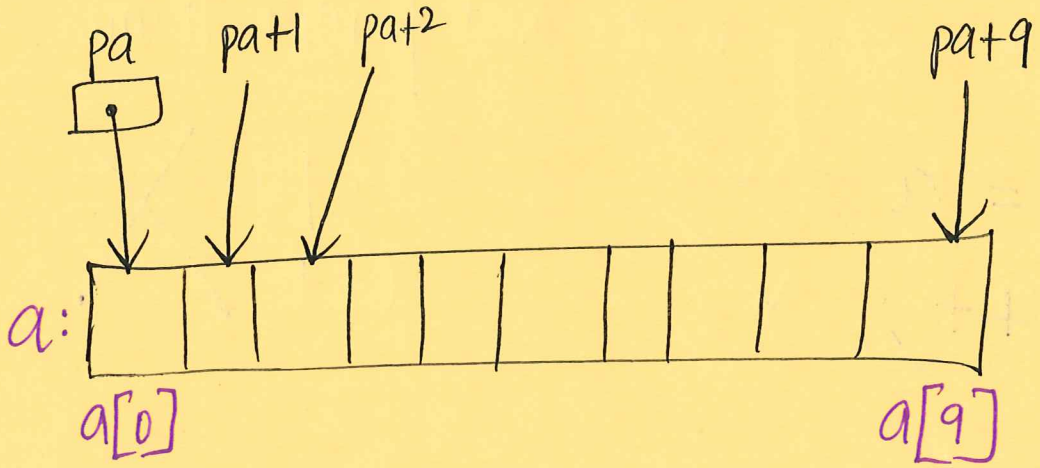



```
int *pa;
pa = &a[0];
```




```
x = *pa;
```





(X) (4)

$$*(a+i) \equiv a[i]$$

Applying & operator on both sides.

$$\&(*(a+i)) \equiv \&(a[i])$$

$$\Rightarrow \quad a+i \equiv a[i]$$

---

if pa is a pointer,

then $pa[i] \equiv *(pa+i)$

(X) ∴ an array-and-index expression $\equiv$ a pointer and offset

* Diff bet array name and pointer names.

pa = a; ✓

pa++; ✓

a = pa; ✗

a++ ✗

* Passing array names to functions.

```
len = mystrlen(a);

int strlen(char *s)
{
    int n;
    for(n=0; *s!='\0'; s++)
        n++;
    return n;
}
```

---

f(&a[2])
and f(a+2)
} pass a part of an array.

```
f(int arr[]) {...}
f(int *arr) {...}
```

NULL pointer

```
int *p = NULL;
```

stdio.h

```
#define NULL 0
```

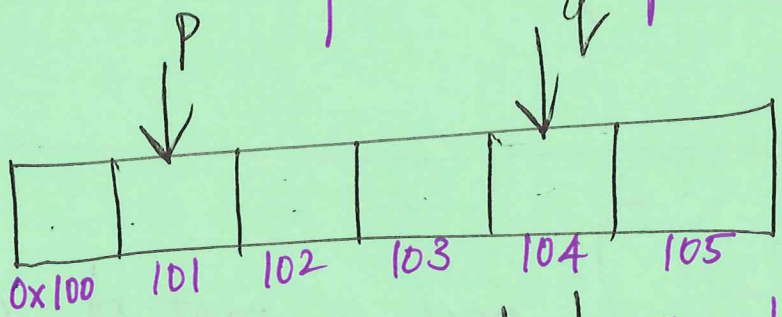
Pointer Arithmetic / Operations

0. Assignment of pointers of same type. eg. p=q

1. Add/Subtract a pointer and an integer

```
p = p+1;    p = p-1;
```

2. Subtract or compare 2 pointers (of same array)



$n = q - p + 1 = 104 - 101 + 1$

$= 4$ elements between p and q (both inclusive).

$p < q$ ✓

3. Assinging or comparing to 0.

```
p=0;  ✓    p=NULL;  (preferred) ✓
```

Illegal pointer operations (7)

(p, q – pointers)

1. p + q ✗
2. p * q ✗
3. p / q ✗
4. p + 7.5 ✗
5. int *pn = &n;

   char ~~int~~ *pc = &c;

   pn = pc; ✗

   pc = (char *) pn; ✓

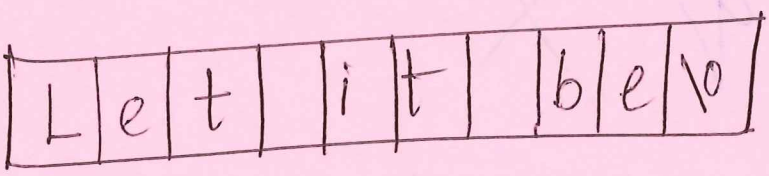
   ⇓

   casting an int * to a char *

* Character pointers

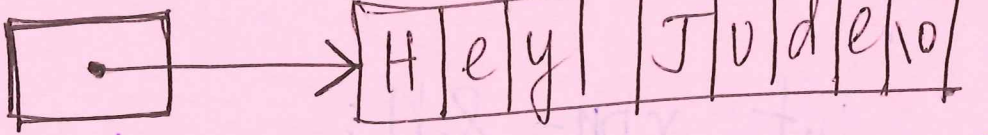
```
char amessage[] = "Let it be";
char *pmessage = "Hey Jude";
```

amessage:



} can be modified

pmessage:



undefined behaviour if modified.

can change pmessage to point to a different string.

String copy

Array version

`s = t;` (only copies pointers).

```
void strcpy(char *s, char *t)
{
    int i;
    i = 0;
    while ((s[i] = t[i]) != '\0')
        i++;
}
```

Pointer version

```
void strcpy(char *s, char *t)
{
    while ((*s = *t) != '\0') {
        s++;
        t++;
    }
}
```

Advanced: void strcpy(char *s, char *t)

```
{
    while ((*s++ = *t++) != '\0')
        ;
}
```

More advanced:

```
while (*s++ = *t++)
    ;
```

To Read in K & R C:

chap 5.1 to 5.5!

Lecture 0 – CS APP: 1

1 – K&R chap 2 (except bitwise operators) → 2.1, 2.2, 2.3, 2.4, 2.7

2 – K&R chap 7 (7.1, 7.2, 7.4, 7.5, 7.7).

Project 0 – due Friday.

---

scanf

BIOS.