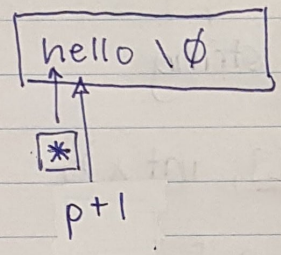
2510 : Lecture 26

Pointer Arithmetic

Motivation :

```
char a [] = "hello";
char *p = a;
```

assume the starting address of the array a is 3276840
What is the content of p (as a number)? 3276840
→ what is p+1 ? = 3276841
→ what is at the address of p+1 ?
Note: the second character ('e') in the array



We can define...
- p+1, p+2, ...
- p-1, p-2, ...
- p++, ++p, p--, --p
- p+=1
- p-=1

Note: p+1 is not a pointer, it's an address
We say p+1 points to 'e'

Standard Idiom to process a String (pointer array)

```
s-string

(const) char *p
        for (p=s; *p != '\0'; p++)
          /* process p */
```

Hilroy

Example:

1) converting a string to all uppercase

```
void str_uppercase (char s[]){
    char *p;
    for(p = s; *p != '\0'; p++){
        *p = toupper (*p);
    }
}
```

2) looking for a character in a string

```
const char * str_find(const char s [], int x){
    const char *;
    for( p = s; *p != '\0'; p++) {
        if (*p == x)
            return p;
    return 0;
```

↳ null pointer

looking for a character in a string v2

```
char * str_find (const char *s, int x) {
    const char *p;
    for (p = s ; *p != '\0' ; p++)
        if (*p == x)
            return (char *) p;
    return 0;
}
```

- returning a char* is more convenient

- a common use case is looking for a character that we want to change

C Library function: char *strchr(const char*, int);

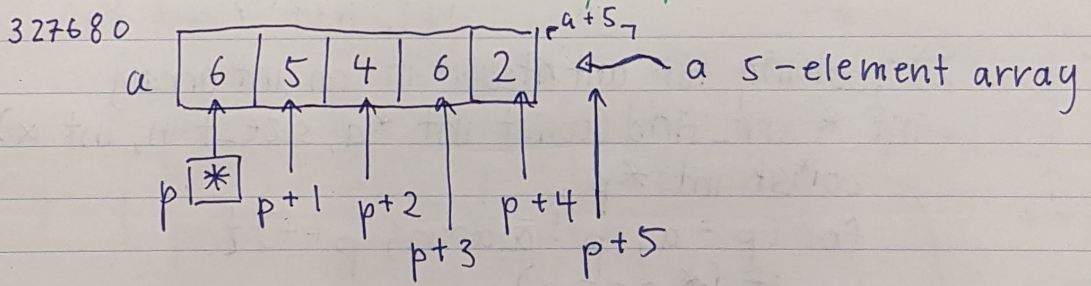
3) Using a pointer to step through an array

```
int a[] = {6, 5, 5, 6, 2};
int *p = a;
```

- assume starting address of a is 3276840
- value of p is also 3276840
- what is p+1?

  Ans: depends on size of an int

  eg. 4 byte ints → value of p+1 = 3276844
  (this is convenient b/c p+1 = address of next element)



Standard idiom to process an array (pointer version)

```
T a[N];
```

T: type; N: some positive integer

```
(const) T *p;
for (p = a; p != a + N; p++)
    /* process *p */
```

Example

1) Summing an integer array

```
int arr_sum (const int *a, size_t n){
    const int *p; int sum = 0;
    for (p = a; p != a + n; p++)
        sum += *p;
    return sum;
}
```

2) looking for an integer in an int array

```
int * arr_find (const int *a, size_t n, int x) {
    const int *p;
    for (p = a; p != a+n; p++) {
        if (*p == x)
            return (int *) p;
    }
    return 0;
}
```

3) squaring the integers in an integer array

```
void arr_square(int *a, size_t n){
    int *p;
    for (p=a; p!=a+n; p++){
        *p *= *p;
}
}
```

▷ *p++, *++p, ++*p, (*p)++

*++p same as *(++p) but (*++)p doesn't make sense
++*p same as ++(*p) but (++*)p doesn't make sense
*p++ * and ++ are on the same precedence level, they associate right to left

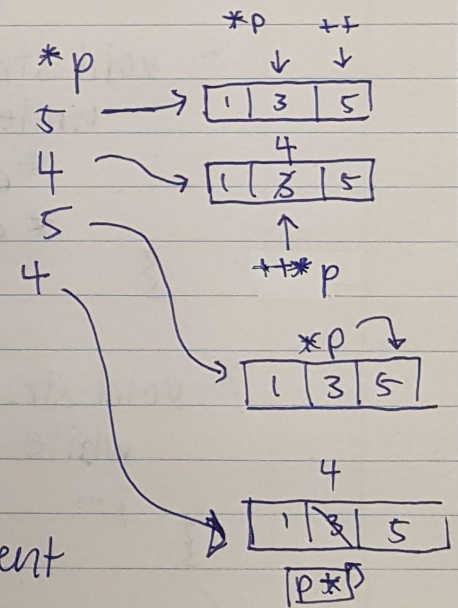
difficult case

∴ *p++ ≡ *(p++)

```
int a[] = {1, 3, 5};
int *p = &a[i];
```

one of the following:

```
printf("%d", *++p);   5
printf("%d", ++*p);   4
printf("%d", *p++);   3
printf("%d", (*p)++); 3
```



*p++ - think of this as processing the current element & then make p point to following element

final material

Hilroy

Example : copying a string

✗
```
void str_copy(char *dest, const char* src){
    char *d;
    const char *s;
    for(d = dest, s = src; *src != '\0'; d++, s++)
        *d = *s;
    *d = '\0';
}
```

- we don't really need d*s as dest + src are local to this function.

✗
```
void str_copy(char *dest, const char *src){
    for (; *src != '\0'; dest++, src++)
        *dest = *src;
    *dest = '\0';
}
```

✗
```
void str_copy (dest *dest, const char *src) {
    while (*src != '\0')
        *dest++ = *src++;
    *dest = '\0';
}
```

✗
```
void str_cpy (char *dest, const char *src){
    while ((*dest++ = *src++) != '\0')
    ;
}
```

✓
```
void str_cpy (char* dest, const char *src){
    while ((*dest++ = *src++))
    ;
}
```

simplicity