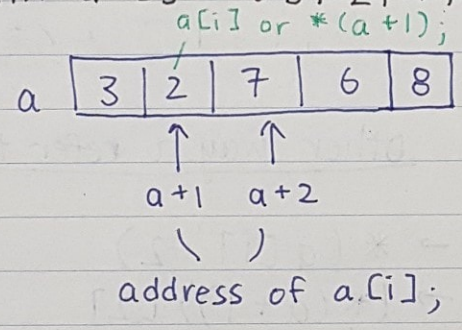
Example : arr_sum

```
int arr_sum(const int *a, size_t n){
    size_t i;
    int    total = 0;
    for (i=0; i<n; i++)
        total += a[i]; /*total += *(a+i); */
    return total;
}
```

- we can use index notation with pointers.

```
int a [] = {3, 2, 7, 6, 8}
```



↑ equivalents of pointer + array notations

$$x[n] \equiv *(x+n) \quad \& x[n] \equiv x+n$$

} x is a pointer/array

▷ Example

1)
```
char a[] = "hello";
char *p = "world";
*(a+1) = p[2];   /* same as: a[1] = p[2] */
*(p+1) = a[2];   /* same as: p[1] = a[2] */
```
changes 'e' to 'r' *

invalid, p points to string constant.

2) a[0] a[1]

```
int a[2][3] = {{1,2,3}, {4,5,6}};
```

② ←——|——→ ①

"a is an array of 2 objects each an array of 3 ints"

a[1][2] is 6 Other ways to refer to this element

→ *(a[1] + 2)
→ (*(a+1))[2]
→ *((*(a+1)) + 2)

Arrays are equivalent to pointer notation but without "notation" they aren't the same

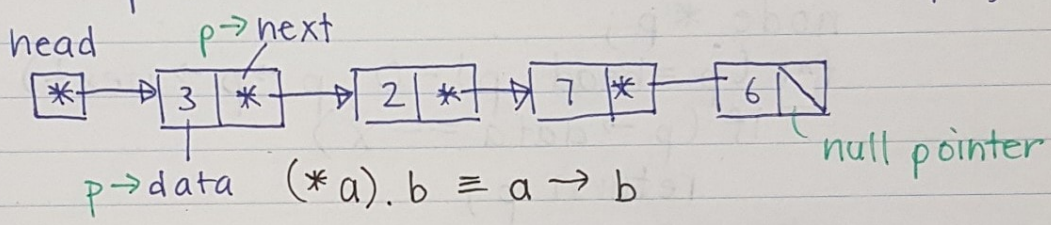
▷ Singly-Linked Lists

↳ linear data structure consisting of nodes each pointing to following node (except for last node)

▷ Example: list of references.

```
typedef struct node node;
struct node {
    int data;
    node *next;
};
```

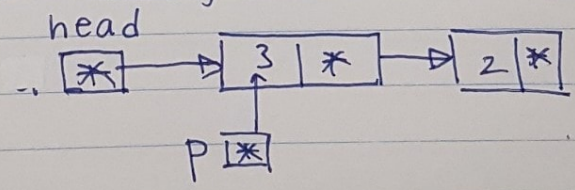
- a linked list is represented by a pointer to its first node (the head pointer)
- this pointer is null if the list is empty



▷ Operations on linked list:

1. Traversing the list
2. Destroying the list
3. Inserting data into list
4. Removing data from list.

Traversing a Linked list



use a pointer to step through each node

Recall: p, q pointers
p = q makes point p to the samething as q.

Hilroy

Standard idiom to traverse a Linked list

```
head - head pointer
node *p;
for (p = head; p != 0; p = p -> next)
   /* standard idiom to process p -> data */
```

▷ Example: looking for a number in the list of ints

```
node * list_find(node * head, int x) {
   node *p;
   for(p = head; p != 0; p = p -> next)
      if (p -> data == x)
         return p;
   return 0;
}
```