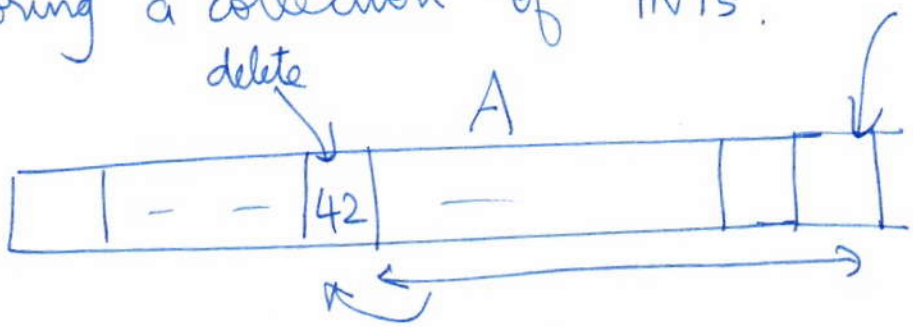
Store words in a DATA STRUCTURE

Given word, FIND: if word is already present.
UPDATE, some data associated with word.
INSERT, the word into the structure.
DELETE, remove word from structure

List : ADT Abstract Data Type

Storing a collection of INTs.



bool find (int val) single for loop

insert (int val) : at the end of the list

↓

insert at A[size]

↳ size++

delete : shifting entries to the "left"

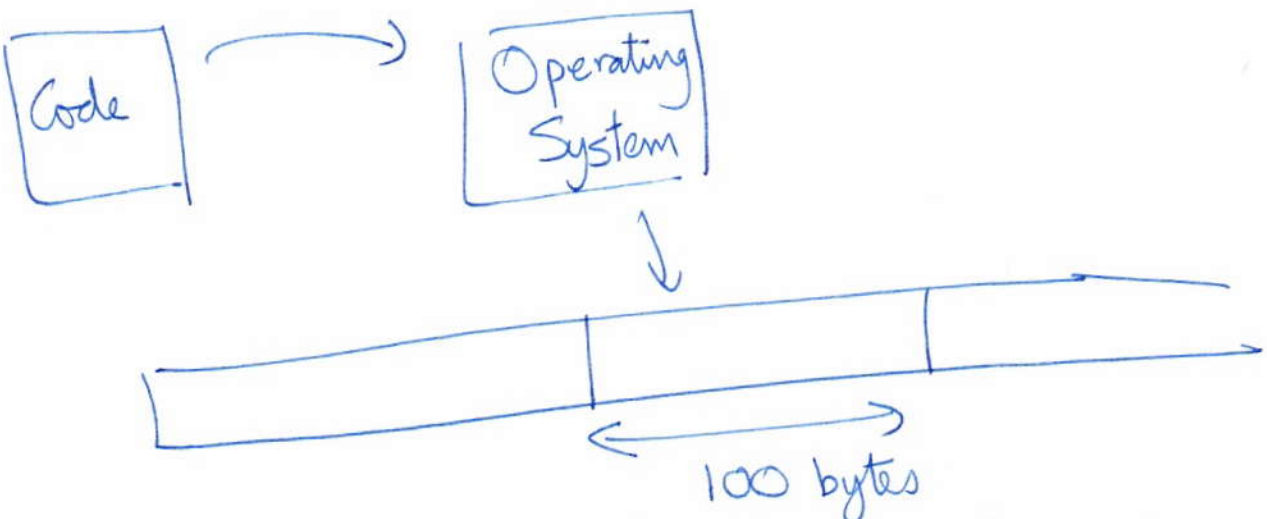
class ~~Linke~~ List (of ints)

{

```
int[]  A;
int    size;
void   List(void)
{
    // allocated memory for the array
    A = new int[ ];
}
```

Wasteful in memory

Give me memory/array of size 100 bytes



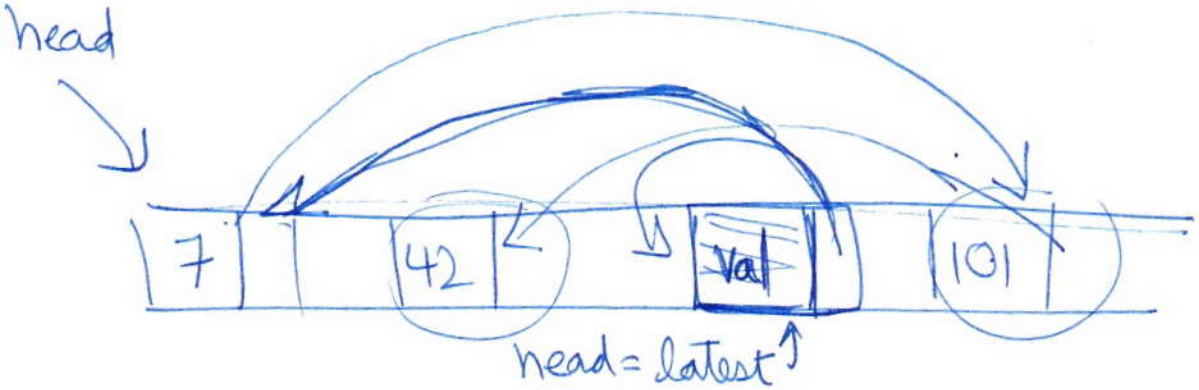
Contiguous for indexing

Linked list is an implementation of the List ADT.

Linked list is a DATA STRUCTURE that implements FIND, INSERT, DELETE.

It allocates exactly the right amount of memory.
This memory does NOT have to be contiguous.

```
class Node
{
    int data;   (in general, this could be an object)
    Node *next;   (pointer to next node)
}
```




```
insert(42);
insert(101);
insert(7);
insert(val);
```

head → [7|•] → [101|•] → [42|\] → NULL

(B) (Or)

(G) No idea

What is (head) pointing to?

(inserts val in some free memory location, and store pointer to previously inserted value)

```
class LinkedList
{
    Node * head; // pointer to first node
                 in list

    void LinkedList()
    {
        head = NULL;
    }
```

```
void insert(val)
{
    Node* latest = new Node();   // creating memory for val
    latest -> data = val;        // storing val
    latest -> next = head;       // linking
    head = latest;               // reset head
    return;
}
```

switch

Will the code throw an error?

Node *find(val)

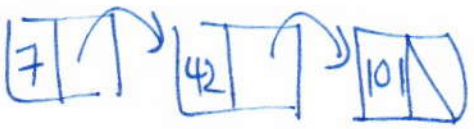
↳ return a pointer to a node with val
return NULL if val is not in list

void print()

↳ prints the list in order

myList.print()

(B) `head = latest;`



~~(Or)~~ `latest -> next = head;`

(G) `head -> next = latest;`

```
void print()
{
    ~~cout <<~~ print(head);
    ~~print(head -> data);~~

    ~~if (head == NULL)~~
        ~~return;~~
}
```

(overloaded) print all nodes in the list beginning from start

```
void print(Node *start)
{
    if (start == NULL)
        return; (or return a new line)
    ~~print(start -> data + " ");~~ cout << start -> dat + " ";
    print(start -> next);
}
```

```
Node * find (int val)
{
    return find(val, head);
}

Node * find(int val, Node* start).
{
    if (start == NULL)
        return NULL;
    if (start -> data == val)
        return start;
    return find(val, start -> next);
}
```