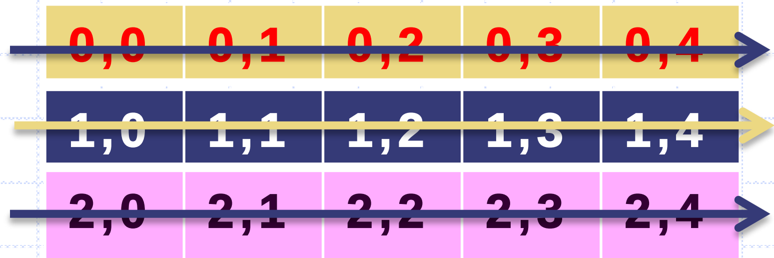
Row Major Layout

mat[3][5]



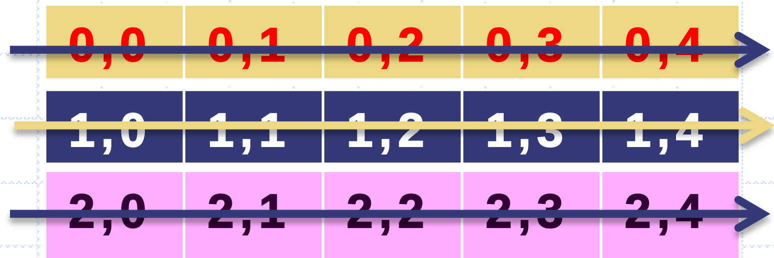
Layout of mat[3][5] in memory

- for a 2D array declared as **mat[M][N]**, cell [i][j] is stored in memory at location **i*N + j** from start of mat.
- for k-D array arr$[N_1][N_2]...[N_k]$, cell $[i_1][i_2]...[i_k]$ will be stored at location

$$i_k + N_k*(i_{k-1} + N_{k-1}*(i_{k-2} + (\dots + N_2*i_1) \dots))$$

Row Major Layout

mat[3][5]



Layout of mat[3][5] in memory

- **About C interpretation:** a = *mat
- *mat = mat[0], *(mat+1) = mat[1], *(mat+2) = mat[2],...... Each of which "stores" the reference to the corresponding row.
- That is, **mat** "points" to the beginning of the array that stores the references to each of the rows.

Array of Strings: Example

- Write a program that reads and displays the name of few cities of India

```
const int ncity = 4;
const int lencity = 10;

int main(){
  char city[ncity][lencity];
  int i;

  for (i=0; i<ncity; i++){
    scanf("%s", city[i]);
  }

  for (i=0; i<ncity; i++){
    printf("%d %s\n", i, city[i]);
  }
  return 0;
}
```

INPUT
Delhi
Mumbai
Kolkata
Chennai

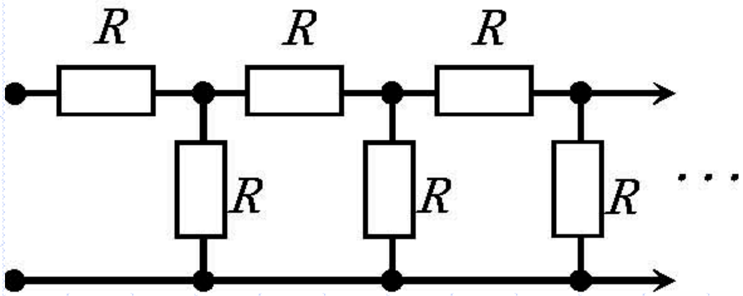
city[0] city[1]

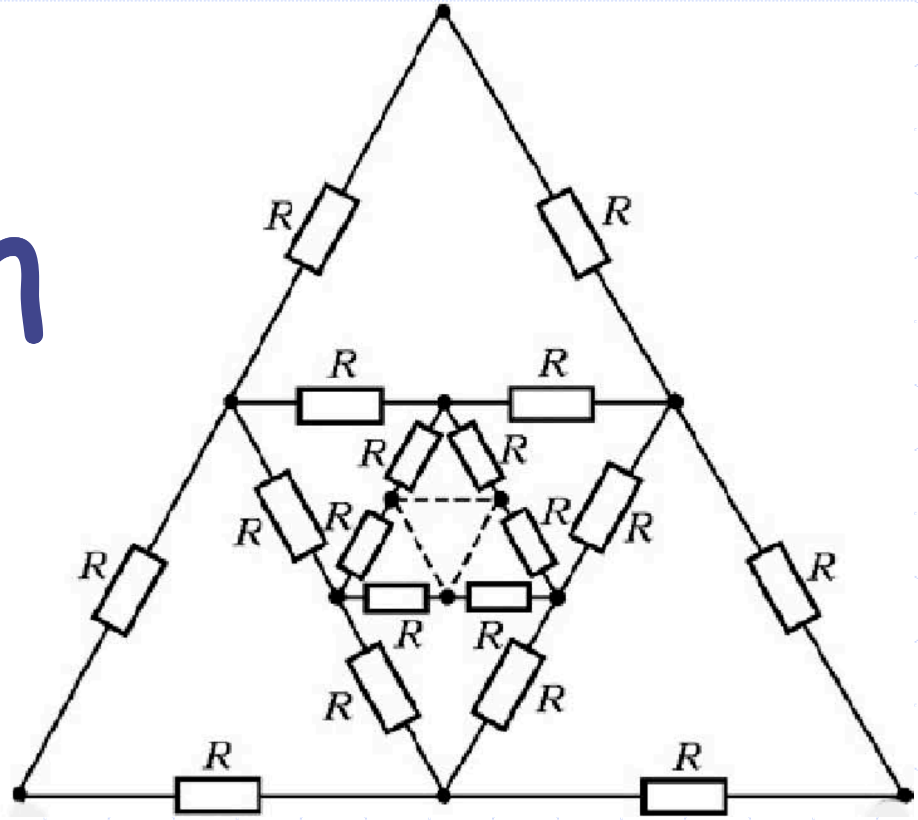
D	e	l	h	i	\0				
M	u	m	b	a	i	\0			
K	o	l	k	a	t	a	\0		
C	h	e	n	n	a	i	\0		

OUTPUT
0 Delhi
1 Mumbai
2 Kolkata
3 Chennai

ESC101: Introduction to Computing

Recursion





Recursion

- A function calling itself, *directly* or *indirectly*, is called a *recursive function.*
  - The phenomenon itself is called recursion
- Examples:
  - Factorial:

```
0! = 1
n! = n * (n-1)!
```

  - Even and Odd:

```
Even(n) = (n == 0) || Odd(n-1)
Odd(n)  = (n != 0) && Even(n-1)
```

Recursive Functions: Properties

- The arguments change between the recursive calls

  5! = 5 * 4! = 5 * 4 * 3! = …

- Change is towards a case for which solution is known (base case)
- There must be one or more base cases

  0! is 1

  Odd(0) is false

  Even(0) is true

Recursion and Induction

When programming recursively, think inductively

- Mathematical induction for the natural numbers
- Structural induction for other recursively-defined types (to be covered later!)

Recursion and Induction

When writing a recursive function,

- Write down a clear, concise *specification* of its behavior,
- Give an *inductive proof* that your code satisfies the specification.

Constructing Recursive functions: Examples

Write a function search(int a[], int n, int key) that performs a sequential search of the array a[0..n-1] of int. Returns 1 if the key is found, otherwise returns 0.

How should we start? We have to think of the function search() in terms of search applied to a smaller array. Don't think in terms of loops…think recursion.

Here's a possibility ….

search(a,n,key)

Base case: If n is 0, then, return 0.

Otherwise: /* n > 0 */

1. compare last item, a[n-1], with key.
2. if a[n-1] == key, return 1.
3. search in array a, up to size n-1.
4. return the result of this "smaller" search.

a

search(a,10,3)

31	4	10	35	59	31	3	25	35	11

Either 3 is a[9]; or search(a,10,3) is same as the result of search for 3 in the array starting at a and of size 9.

```
int search(int a[], int n, int key) {
    if (n==0) return 0;
    if (a[n-1] == key)  return 1;
    return search(a,n-1,key);
}
```

Let us do a quick trace.

a

E.g., (0) search(a,5,10)

31	4	10	35	59

a[4] is 59, not 10. call search(a,4,10)

a

(1) search(a,4,10)

31	4	10	35	59

a[3] is 35, calls search(a,3,10)

a

(2) search(a,3,10)

31	4	10	35	59

a[2] is 10, return 1

```
int search(int a[], int n, int key) {
    if (n==0) return 0;
    if (a[n-1] == key)  return 1;
    return search(a,n-1,key);
}
```

Let us do another quick trace.

E.g., (0) search(a,5,3)

a

31	4	10	35	59

a[4] is 59, not 3. call search(a,4,3)

(1) search(a,4,3)

a

31	4	10	35	59

a[3] is 35, calls search(a,3,3)

(2) search(a,3,3)

a

31	4	10	35	59

a[2] is 10, calls search(a,2,3)

(3) search(a,2,3)

a

31	4	10	35	59

a[1] is 4, calls search(a,1,3)

(4) search(a,1,3)

a

31	4	10	35	59

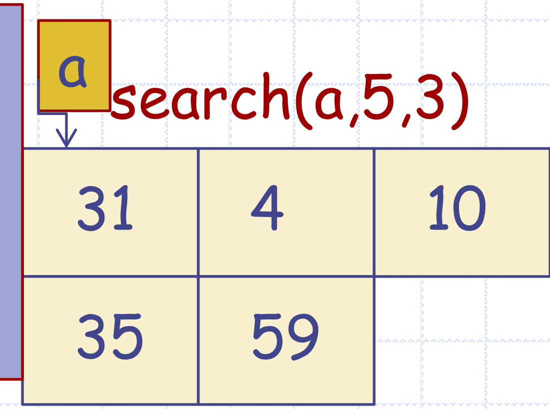
a[0] is 31, calls search(a,0,3)

a

31	4	10	35	59

(5) search(a,0,3) returns 0

```
1. int search(int a[], int n, int key) {
2.     if (n==0) return 0;
3.     if (a[n-1] == key)  return 1;
4.     return search(a,n-1,key);
5. }
```



Stack

function	called by	return address	return value
search(a,5,3)	main()	---	
search(a,4,3)	search(a,5,3)	search.5	
search(a,3,3)	search(a,4,3)	search.4	
search(a,2,3)	search(a,3,3)	search.3	
search(a,1,3)	search(a,2,3)	search.2	
search(a,0,3)	search(a,1,3)	search.1	

recursion exits here

A state of the stack

```
1. int search(int a[], int n, int key) {
2.     if (n==0) return 0;
3.     if (a[n-1] == key)  return 1;
4.     return search(a,n-1,key);
5. }
```

a

search(a,5,3)

31	4	10
35	59	

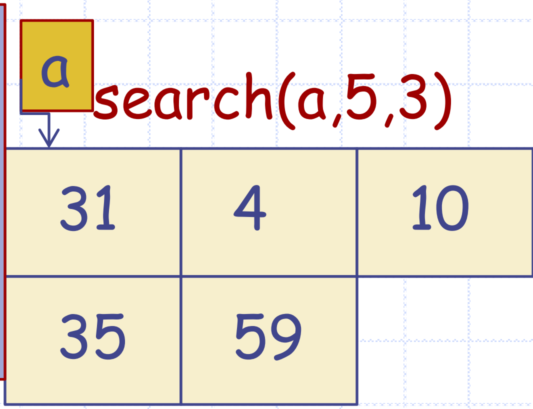
Stack

function	called by	return address	return value
search(a,5,3)	main()	---	
search(a,4,3)	search(a,5,3)	search.5	
search(a,3,3)	search(a,4,3)	search.4	
search(a,2,3)	search(a,3,3)	search.3	
search(a,1,3)	search(a,2,3)	search.2	
search(a,0,3)	search(a,1,3)	search.1	0

recursion exits here

A state of the stack

```
1. int search(int a[], int n, int key) {
2.     if (n==0) return 0;
3.     if (a[n-1] == key)  return 1;
4.     return search(a,n-1,key);
5. }
```

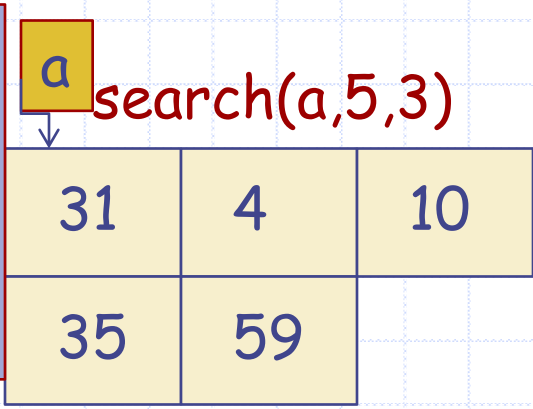


Stack

function	called by	return address	return value
search(a,5,3)	main()	---	
search(a,4,3)	search(a,5,3)	search.5	
search(a,3,3)	search(a,4,3)	search.4	
search(a,2,3)	search(a,3,3)	search.3	
search(a,1,3)	search(a,2,3)	search.2	0

state of the stack

```
1. int search(int a[], int n, int key) {
2.     if (n==0) return 0;
3.     if (a[n-1] == key)  return 1;
4.     return search(a,n-1,key);
5. }
```



Stack

function	called by	return address	return value
search(a,5,3)	main()	---	
search(a,4,3)	search(a,5,3)	search.5	
search(a,3,3)	search(a,4,3)	search.4	
search(a,2,3)	search(a,3,3)	search.3	0

A state of the stack

```
1.  int search(int a[], int n, int key) {
2.      if (n==0) return 0;
3.      if (a[n-1] == key)  return 1;
4.      return search(a,n-1,key);
5.  }
```

a

search(a,5,3)

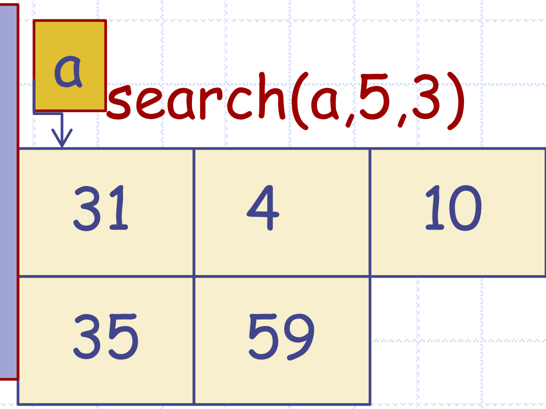
31	4	10
35	59	

Stack

function	called by	return address	return value
search(a,5,3)	main()	---	
search(a,4,3)	search(a,5,3)	search.5	0

A state of the stack

```c
int search(int a[], int n, int key) {
    if (n==0) return 0;
    if (a[n-1] == key)  return 1;
    return search(a,n-1,key);
}
```

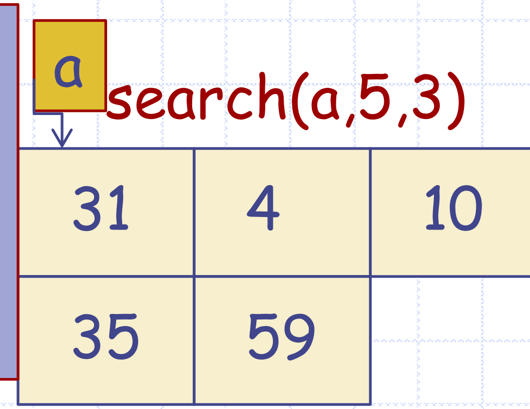


Stack

function	called by	return address	return value
search(a,5,3)	main()	---	0

A state of the stack

```
int search(int a[], int n, int key) {
    if (n==0) return 0;
    if (a[n-1] == key)  return 1;
    return search(a,n-1,key);
}
```



search(a,5,3) returns 0. Recursion call stack terminates.

Searching in an Array

- We can have other recursive formulations
- Search1: search (a, start, end, key)
  - Search key between a[start]...a[end]

```
if start > end, return 0;
if a[start] == key, return 1;
return search(a, start+1, end, key);
```

Searching in an Array

- One more recursive formulation
- Search2: search (a, start, end, key)
  - Search key between a[start]…a[end]

```
if start > end, return 0;
mid = (start + end)/2 ;
if a[mid]==key, return 1;
return search(a, start, mid-1, key)
          || search(a, mid+1, end, key);
```