ESC101: Introduction to Computing

Multi dimensional Arrays

Practice Problem

- Each course given as a string.
- Each course has with it its pre-requsite course listed (NULL if no pre-requisite)
- Input: List of 5 courses with its pre-requisite
- Output: A sequence of courses to be followed (if CS201 and CS210 both are possible, CS201 should be output before CS210)

Input
ESC101 NULL
CS210 ESC101
CS345 CS210
CS340 CS201
CS201 ESC101

Output
ESC10 CS201 CS210 CS340 CS345

Approach

- Assume we can use a sort routine to sort list of courses and pre-requisites
- Take str="NULL" as first pre-requisite and an array seq to store the sequence number of each course and prereq=0
- The first course that has NULL as pre-requisite gets the first sequence number in seq, the next course gets next number
- Now take the first number in seq as prereq and check the list of courses and again assign sequence numbers

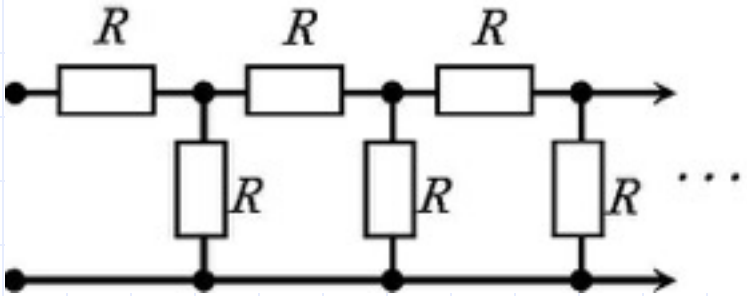
```
void order_courses( char course[5][100], char prereq[5][100])
{
   char str[100]="NULL";
   int cnt=1;
   //looping over prereq with i
   for( int i=1; i<5; i++)
   {
      //looping over courses to check if i is a prereq
      for(int j=0; j<5; j++)
      {
         if( strcmp(prereq[j],str) == 0 )
            seq[j] = cnt++;
      }
      //obtaining next prereq
      for(int j=0; j<5; j++)
         if(seq[j] == i)
            strcpy(str, course[j]);
   }
}
```

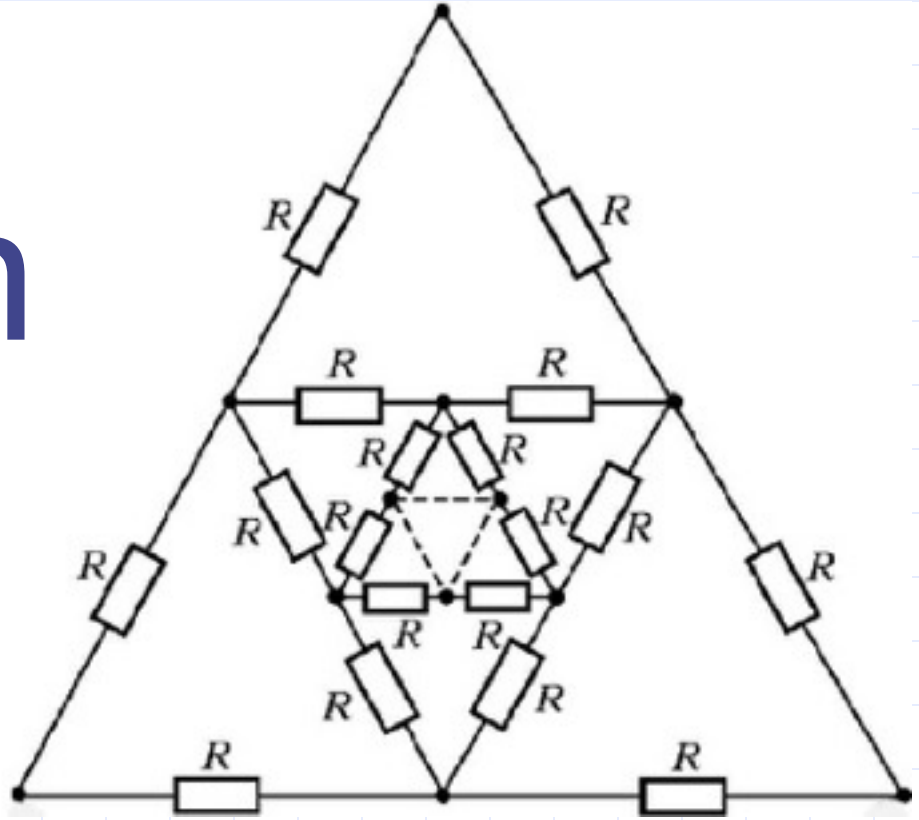
```
#include <stdio.h>
#include <string.h>

int seq[5] = {0};
void swap( char s1[100], char p1[100], char s2[100], char p2[100])
void sort_courses( char crs[5][100], char prq[5][100] );
void order_courses( char crs[5][100], char prq[5][100]);
int main()
{
   char course[5][100];
   char prereq[5][100];

   for(int i=0; i<5; i++)
      scanf("%s %s",course[i], prereq[i] );

   sort_courses( course, prereq );
   order_courses( course, prereq );
    for(int i=1; i<=5; i++)
      for(int j=0; j<5; j++)
         if(seq[j] == i)
            printf("%s\n",course[j]);
    return 0;
}
```

```
void swap( char s1[100], char p1[100], char s2[100], char
p2[100])
{
   char str[100];
   strcpy( str, s1);
   strcpy( s1, s2 );
   strcpy( s2, str );
   strcpy( str, p1);
   strcpy( p1, p2 );
   strcpy( p2, str );
}
void sort_courses( char courses[5][100], char prereq[5][100] )
{
   for(int i=0; i<5; i++)
   {
      for(int j=i+1; j<5; j++)
      {
         if(strcmp(courses[i],courses[j])>0)
            swap(courses[i], prereq[i], courses[j], prereq[j]);
      }
   }
}
```

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 behaviour.
- Give an inductive proof that your code satisfies the specification.

Factorial of a number

Task: Given n, compute n!

```
Function factorial(int n)
//base case
if (n == 0) return 1;
//inductive step
else
 return n*factorial(n-1);
```

Factorial of a number

```
#include <stdio.h>

int factorial( int n)
{
   if(n<=0)
    return 1;
   else
    return n*factorial(n-1);
}

int main()
{
   int n;
    scanf("%d",&n);
    printf("n! is%d\n",factorial(n) );
    return 0;
}
```

Consider search problem

Task: Given a key, return 1 if it is in an integer array or -1 if not

```
Function find_key(int a[], int key, int n)
for(i = 0; i< n; i++)
    if(a[i] == key)
        return 1
return -1;
```

```c
#include <stdio.h>

int find_key(int arr[], int key, int n);
int main()
{
   int arr[]={100,10,4,20,45,56,72,43,33,93};
   int key;
   scanf("%d",&key);
   printf("%d\n",find_key(arr,key, 10) );
   return 0;
}

int find_key(int arr[], int key, int n)
{
   for(int i=n-1; i>=0; i--)
   {
      if(arr[i] == key)
         return 1;
   }
   return -1;
}
```

Recursive search

Task: Given a key, return 1 if it is in an integer array or -1 if not

```
Function find_key(int a[], int key, int n)
if (n == 0)
      return -1;
if (a[n-1] == key)
      return 1;
else
     return find_key(a,key, n-1);
```

search(a[],n,key)

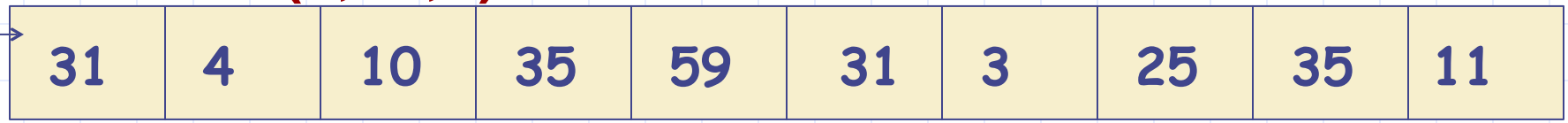
Base case: If n is 0, then, return **0**.

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)



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.

```
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. }
```

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

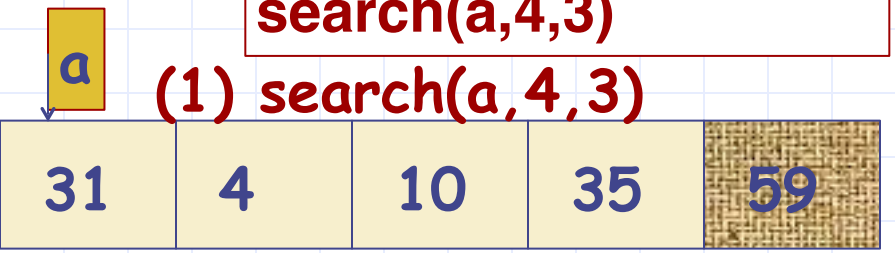
```
1.  int search(int a[], int n, int key) {
2.      if (n==0) return 0;
3.      if (a[n-1] == key)  return 1;
4.      return search(a,n-1,key);
5.  }
```

Let us do another quick trace.

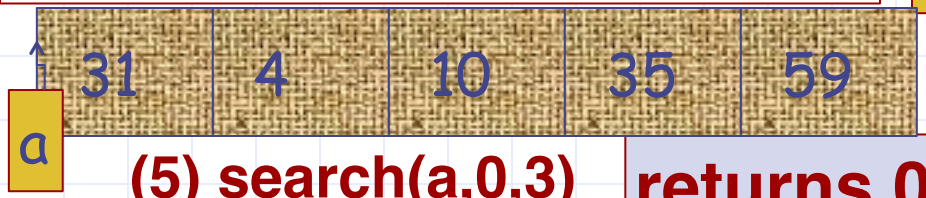
E.g., (0) search(a,5,3)

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

(1) search(a,4,3)

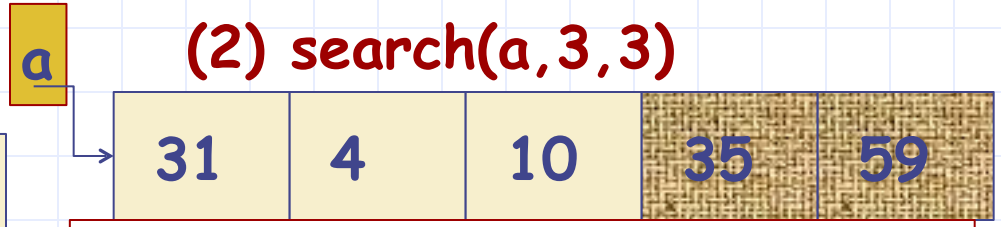


a[3] is 35, calls search(a,3,3)



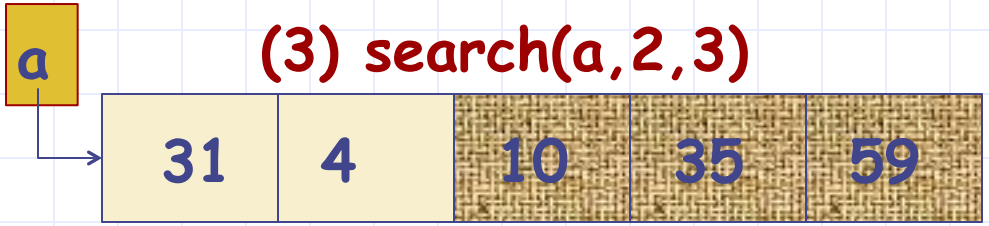
(5) search(a,0,3) **returns 0**

(2) search(a,3,3)



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

(3) search(a,2,3)

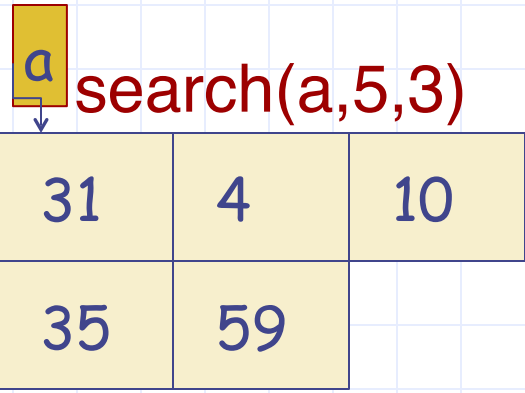


a[1] is 4, calls search(a,1,3)

(4) search(a,1,3)

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

```
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. }
```



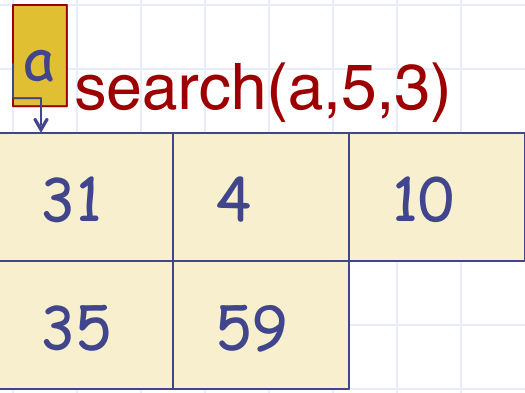
Stack

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. }
```



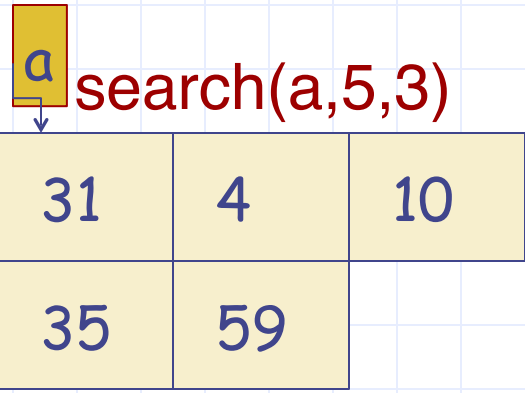
Stack

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.  }
```

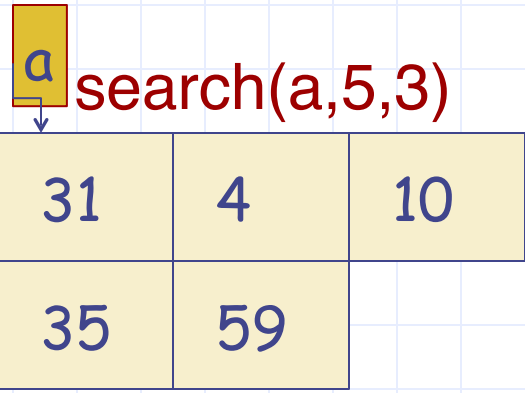


Stack

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

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.  }
```

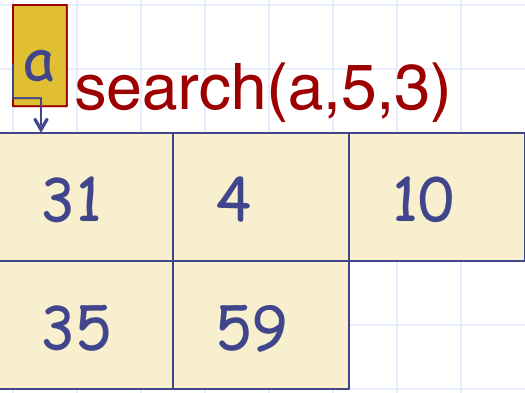


Stack

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	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. }
```

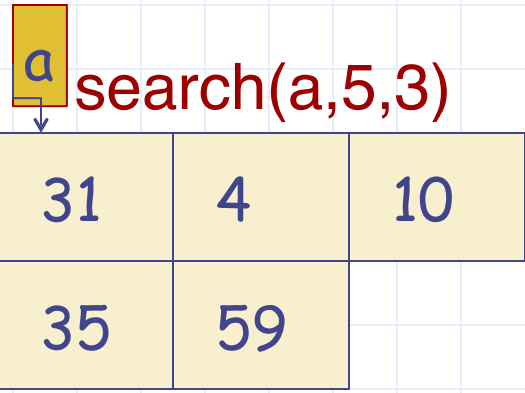


Stack

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

```
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. }
```

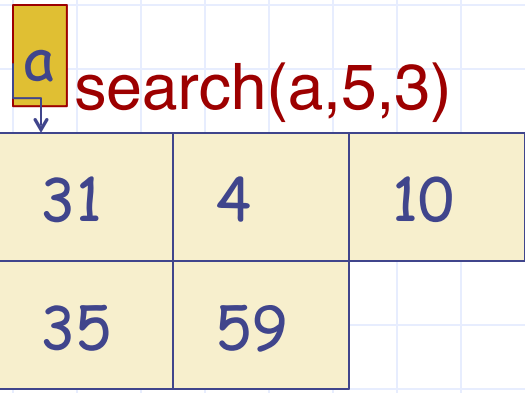


Stack

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



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);
```

Example 2: In-place reversing an array

Write a function reverse(int a[], int n) that reverses the values contained in the first n indices of a[]. That is, a[0] and a[n-1] are exchanged, a[1] and a[n-2] are exchanged, and so on.

reverse (a,n): formulating the problem recursively

Basic idea:

1. if n is 0 or 1, return. Nothing to reverse.
2. Otherwise,
   a) exchange a[0] with a[n-1].
   b) call reverse on array starting at position 1 and of size n-2.

Let's write this…

```
void reverse(int a[], int start, int end) {
   if (start==end || end-start==1 ) return ;
   else {
         swap(a,start,end-1);
         reverse(a,start+1, end-1);
   }
}

void swap( int a[], int n1, int n2)
{
   int tmp=a[n1];
   a[n1] = a[n2];
   a[n2] = tmp;
}

int main()
{
   int arr[]={100,10,4,20,45,56,72,43,33,93};
   for( int i=0; i<10; i++)
      printf("%d ",arr[i]);
   printf("\n");
   reverse(arr, 0,10);
   for( int i=0; i<10; i++)
      printf("%d ",arr[i]);
   printf("\n");
   return 0;
}
```