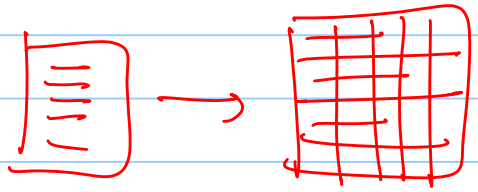
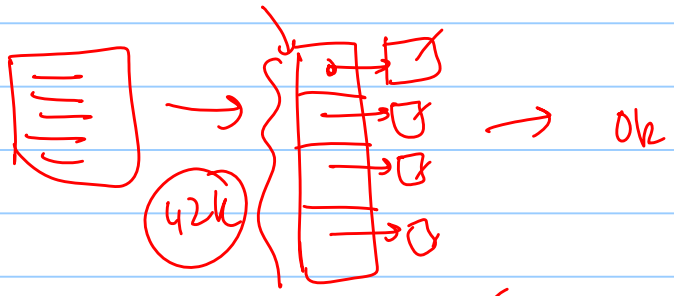
Lab3

Note Title 9/15/2009

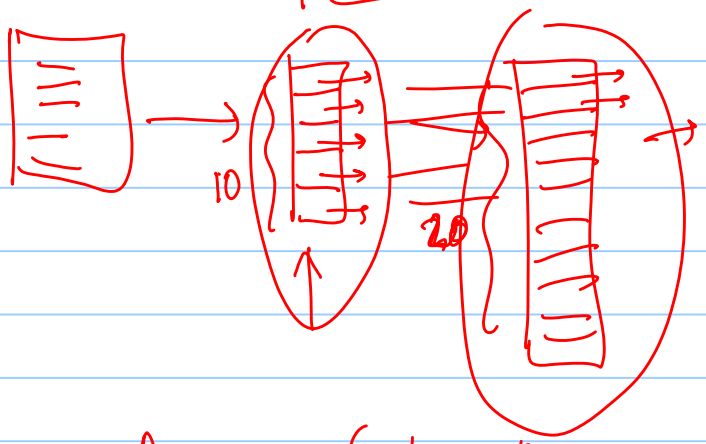
Lab1



Lab2



Lab3



Complexity

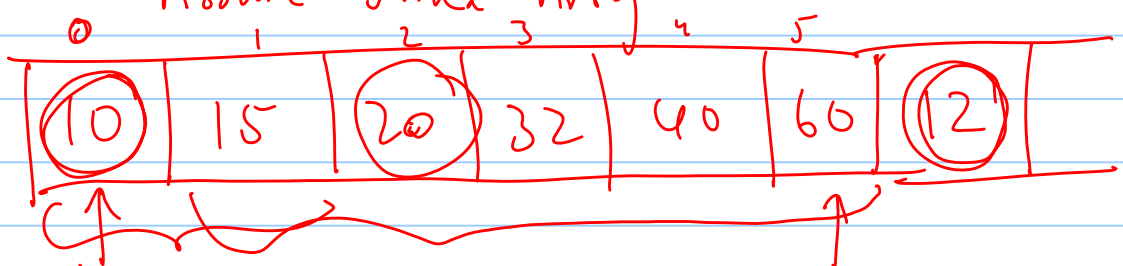
big O

Linear Search O(n)

binary Search O(log n)

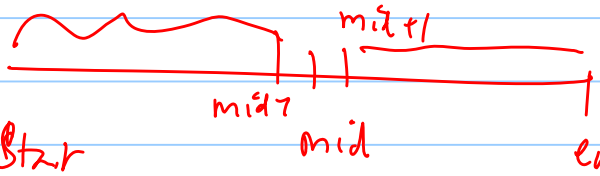
$n = 10^6$ $\log_2 10^6 \simeq 20$

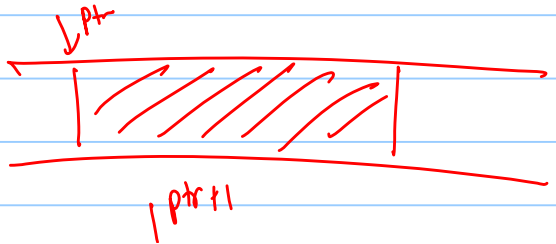
Assume Sorted Array



begin	end	mid
0	~~8~~	2
0	1	0
1	~~1~~	1

1 > 0 → exit





Lecture 07
Pointers, *, ** and ***

In this lecture

- Revisit pointer
- Pointer arithmetic
- Passing a pointer to a function
- ** the address of a *
- *** the address of a **
- Dealing with ***
- Further readings
- Exercises

Revisiting pointers

A pointer is an address in the memory. Once the address of a memory location is provided to a function, a function can make changes to the actual content of the location. For example,

```
int x=10;
foo(&x);
```

where foo is defined as

```
void foo(int* ptr){
    (*ptr)++;
}
```

Will actually increase the value of x by 1. Note that ptr is dereferenced first, that is (*ptr) before being incremented.

Question: What happens if *ptr++ is written instead of (*ptr)++ ?

Pointer Arithmetic

A pointers can be added and subtracted. For example, if ptr1 and ptr2 are pointers (of the same type) then we can define the following.

1. Ptr1 + n defines the address of a location that is n locations from the ptr1. For example, if ptr1 is an int*, then ptr1+n defines the address of ptr1[n]

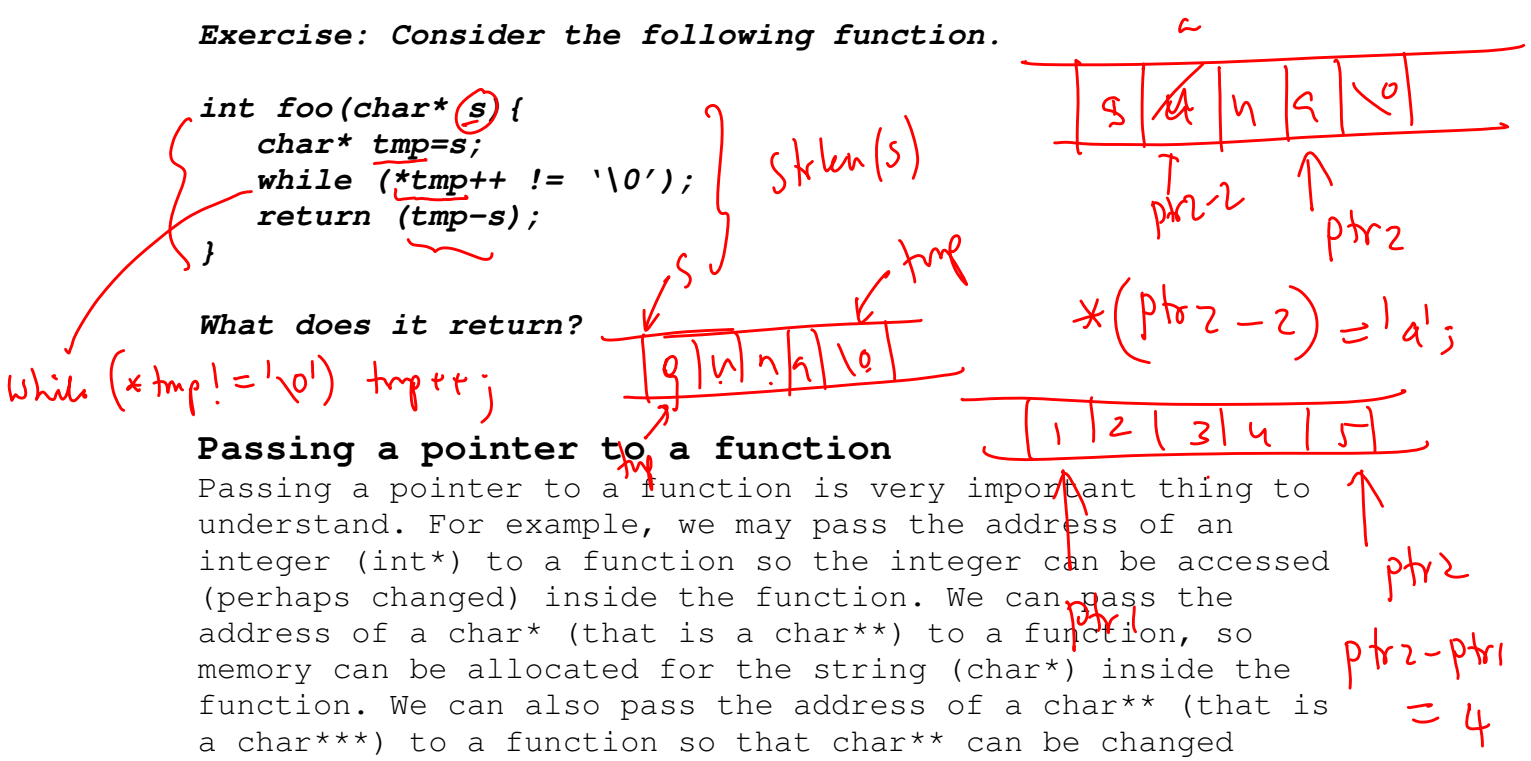
2. ptr2 – n defines the address of a location that is n locations before ptr2. For example, if ptr2 is a char*, then ptr2–n defines the address of the nth character from ptr2.

3. If ptr1 and ptr2 are both int*'s then ptr2–ptr1 defines the number of integers between ptr1 and ptr2

Exercise: Consider the following function.

```
int foo(char* s) {
    char* tmp=s;
    while (*tmp++ != '\0');
    return (tmp-s);
}
```

What does it return?

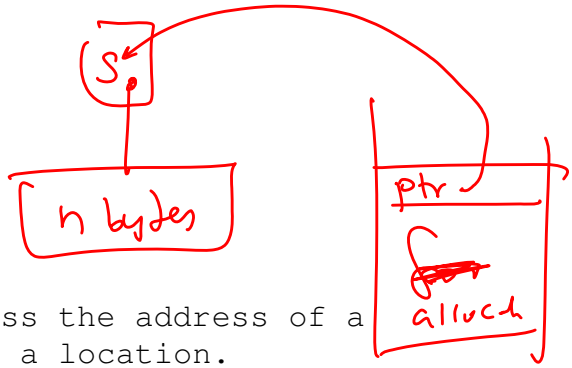


Passing a pointer to a function

Passing a pointer to a function is very important thing to understand. For example, we may pass the address of an integer (int*) to a function so the integer can be accessed (perhaps changed) inside the function. We can pass the address of a char* (that is a char**) to a function, so memory can be allocated for the string (char*) inside the function. We can also pass the address of a char** (that is a char***) to a function so that char** can be changed inside the function.

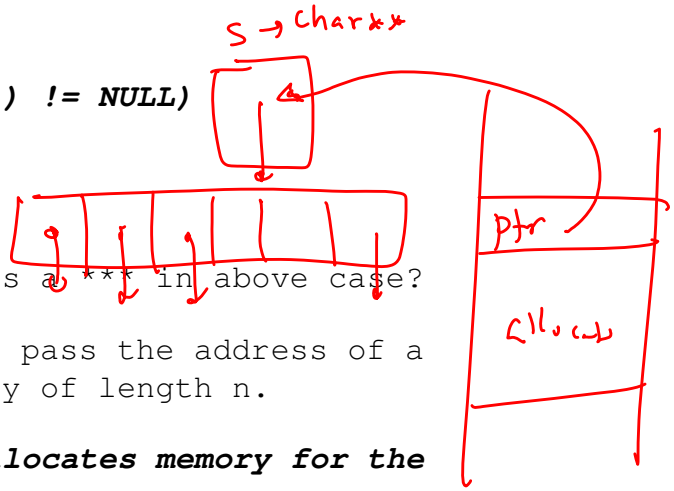
Example 1: This example shows how to pass the address of a char* to assign a string to a location.

```
char* s = NULL; /* this does not allocate memory for the string*/
allocate(&s, n); /*call the function to allocate memory of n bytes for s*/

int allocate(char** ptr, int n){
   if ((*ptr=malloc(n)) != NULL)
         return 0;
   return 1;
}
```



Example 2: This example shows how to pass the address of a char** to assign an array of char*'s to a location.

```
char** s = NULL; /* this does not allocate memory for the array of strings */
allocate(&s, n); /*call the function to allocate memory of for an array of char*'s */

int allocate(char*** ptr, int n){
   if ((*ptr=malloc(n*sizeof(char*))) != NULL)
         return 0;
   return 1;
}
```

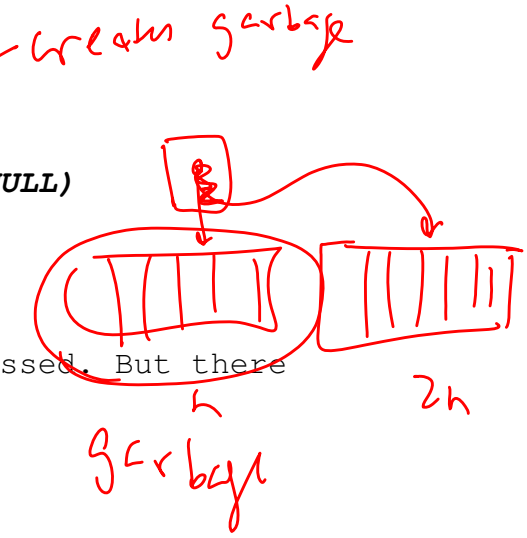


Question: Why is that we have to pass a *** in above case?

Example 3: This example shows how to pass the address of a char** to double the size of an array of length n.

```
char** s = NULL; /* this does not allocates memory for the array of strings */

allocate(&s,n); /* allocate as defined in example 2. Now this allocates an array of n char'*s */

doubleArray(&s, n); /*call the function to double the size of the array */
```

```
int doubleArray(char*** ptr, int n){
   if ((*ptr=malloc(2*n*sizeof(char*))) != NULL)
         return 0;
   return 1;
}
```



Question: This doubles the array that was passed. But there are problems. What are they?

More examples from previous notes

Example 1

Write a function that takes the name of a file (char*) that contains ints, an array of ints and the address of a variable count and reads the file into the array. Assume that the array has enough space to hold the file. count should be updated to the number of entries in the file.

Answer:

```
int foo(char* filename, int A[], int* countptr){
    FILE* fp=NULL;
    int num=0;
    if ((fp=fopen(filename,"r")) != NULL){
      while (fscanf(fp,"%d",&num)>0)
          { A[*countptr]= num;
            *countptr += 1;
          }
      return 0;
    }
    else return 1;
```

Insert Discussion from lecture

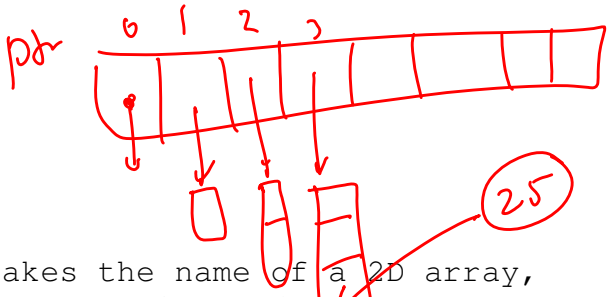
Example 2

Consider the following declaration.

int matrix;

Write a function matrixAllocate that takes two integers, m and n and allocate an m by n block of memory.

```
int matrixAllocate(int*** Mptr, int n, int m){
     *Mptr =  (int**)malloc(m*sizeof(int*));
     int i=0;
     for (i=0;i<m;i++)
          (*Mptr)[i] = malloc(n*sizeof(int));
}
```



insert Discussion from lecture

Example 3

Write a C function swap that takes the name of a 2D array, num rows, num columns, and two values i and j and swap the two rows i and j. All error checking must be done.

```
int swap(int M[m][n], int i, int j){
```

int m,n;

int tmp[n] = M[i];

M[i] = M[j];

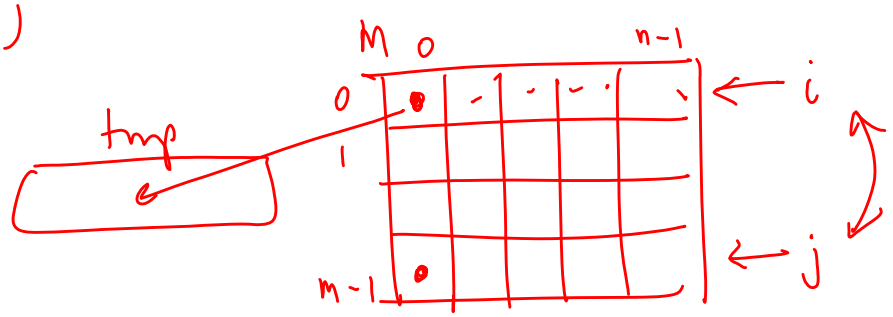
M[j] = tmp;

*(ptr[3]+2) = 25

or

ptr[3][2] = 25

```
}
```



Insert Discussion from lecture

Further Readings

See K & R sections 5.7-5.9

Exercises

[1] Write a function freeAll(char* A[],int n) that takes an array of char*'s and delete all memory associated with A

[2] Learn more about valgrind, a tool to check memory leaks. Type: man valgrind