Question 1 (3 pts) In C when an array is passed to a function, it is 'decayed' into a pointer. Specifically, it is the single starting address of the array, not the array itself, that is passed to the function. List one advantage of this mechanism, and one disadvantage of it.

Question 2 (1 pt) Suppose you are going to write an ANSI C program that takes user inputs line by line. Each input line contains variable lengths in the range of 1~200 characters. You need to maintain a table of these input lines, so that at the end you can sort the input lines.

1) **What are the two possible data structures in C that can store the inputs?**

2) **For this problem, which is the preferred data structure here?**

3) **Discuss briefly the potential advantage(s) of using this data structure over using the other one?**

For Q1

Advantages: Array is passed efficiently; easy passing of sub-arrays (which allows recursion as shown above).
Disadvantages: need to provide the length info of array to function, either by passing leng info as extra argument, or, put a terminator token at the end of array.

For Q2

1) What are the two possible data structures in C that can store the inputs?
2D array char[][], array of pointers char * []
2) For this problem, which is the preferred data structure here?

Array of pointers
3) Discuss briefly the potential advantage(s) of using this data structure over using the other one?

Since input has variable length, this potentially save space.
Allow swapping of info by exchange addresses, rather than move data.

In the linked list implementations in labtest 2 and lab, to make the code easier, we declare the `head` to be a global variable. An alternative way is to declare `head` as a local variable and pass it as a function parameter.

Write a function `return_type insertFirst(struct node* head, char d)` which inserts in the beginning of the list a new node with data `d`. You need to define the return type of the function. You also needs to call this function to insert nodes with data `'A'`and `'E'` into the list.

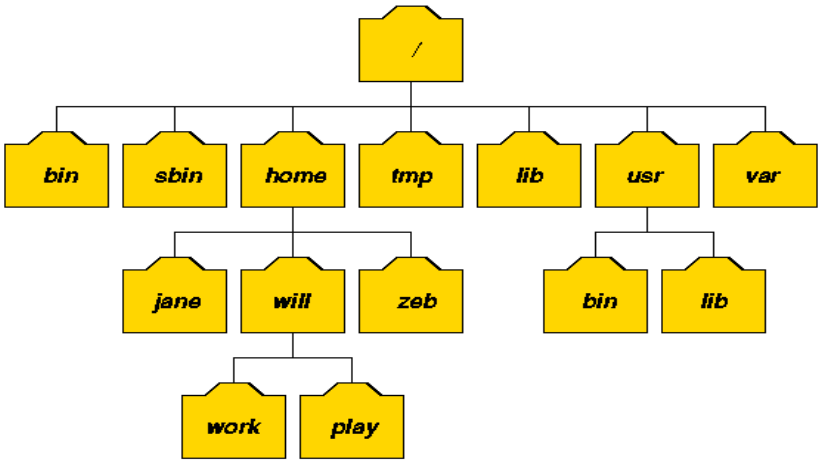
```
#include <stdio.h>

struct node {

   char data;
   struct node *next;
};

/************* DO NOT ADD MORE GLOBAL VARIABLES *************/

/*Define your function below:*/

  struct node *  insertFirst (struct node * list_head, char d){

  /***** ADD YOUR CODE HERE *****/


   struct node * newP = malloc (sizeof(struct node));

   newP -> data = d;

   newP -> next = list_head;

   list_head = newP;


   return list_head;

 }


main(){

   struct node * head;

   /* call your function (twice) to insert nodes of data 'A' 'E' into list */

   head = insertFirst (head, 'A');

   head = insertFirst (head, 'E');

}
```

Question 7 short answers

What are the four types of files maintained by the Unix file system?

Regular file; directory; device file; link file

Consider the following file system structure. Assume the current working directory is `/home/will/play`. Suppose there is a file called `foo` in `/home/will/work`. Using a **relative path,** and without changing the current working directoy, copy `foo` to `/usr/bin`



`cp ../work/foo  ../../../usr/bin/` or

`cp ../work/foo  ../../../usr/bin/foo`

2.11 Given a text file `classlist`, one student per line,

a) complete the `echo` utility so that the output is `class 2031 has x students` where `x` is the number of students in the class

```
$ echo  "class 2031 has      `wc -l classlist`   students"
```

b) complete the `echo` utility so that the output is `class 2031 has x students with name Wang or Wong` where `x` is the number of students whose (last name) is `Wang`.

```
$ echo  "class 2031 has      `grep Wang classlist | wc -l`
students with name Wang"
```

What does this utility do?

```
find ../  -mtime -3  -exec cp {} {}.bak \;
```

find all the files that has been modified within the 3 days, and then copy to file with sufffx .bak

9.13 Given a text file `foo`, use one Unix utility/command to find all the lines of `foo` that:

a) ends with a character that is not `a` nor `b` nor `c`, followed by `nd`

```
grep  [^a-c]nd$ foo
```

The **who** utility displays who is logged on. An example is shown below:

```
john     pts/10       Jul 29 14:21 (deepthought.cs.yorku.ca)
alice    pts/13       Aug  1 21:50 (bas1-toronto07-1176321369.dsl.bell.ca)
xijunw   pts/15       Jul 30 00:12 (grad335-287.resnet.yorku.ca)
rouhan   pts/16       Aug  1 15:10 (cpe0e.net.cable.rogers.com)
midas    pts/24       Jul 22 15:17 (monster.cs.yorku.ca)
gaodon   pts/26       Jul 31 16:01 (65.95.93.10)
tom      pts/32       Jul 31 11:01 (65.95.93.10)
peter    pts/37       Jul 27 10:09 (65.95.93.10)
```

Give **one** line of Unix commands/utilities, possibly connected by pipes but **without** using redirection (> symbol) and grouping (; symbol), to find and displays who is currently logged on, and sort and displays it based on the log on time. The sorted results of the above list is shown below

```
midas    pts/24       Jul 22 15:17 (monster.cs.yorku.ca)
peter    pts/37       Jul 27 10:09 (65.95.93.10)
john     pts/10       Jul 29 14:21 (deepthought.cs.yorku.ca)
xijunw   pts/15       Jul 30 00:12 (grad335-287.resnet.yorku.ca)
tom      pts/32       Jul 31 11:01 (65.95.93.10)
gaodon   pts/26       Jul 31 16:01 (65.95.93.10)
rouhan   pts/16       Aug  1 15:10 (cpe0e.net.cable.rogers.com)
alice    pts/13       Aug  1 21:50 (bas1-toronto07-1176321369.dsl.bell.ca)
```

Your solution should display on stdout **both** the original list and (followed by) the sorted list

```
who | sort -k 5
```

9.14 Given a text file `foo`, which contains 20 lines, and the content is

```
This is line 1
This is line 2
```

…..

```
This is line 20
```

what is the output of the following utility?

```
$ tail  -5  foo  |  head -3
```

This is line 16
This is line 17
This is line 18

9.11 Assume the current directory contains some files and (sub)directories.

a) without using `sort` utility and pipe, use one Unix utility to list the files/subdirectories in the current directory line by line, sorted by the modification time (in ascending or descending order).

ls -lt

b) without using `sort` utility and pipe, use one Unix utility to list the files/subdirectories in the current directory line by line, sorted by the file size (in ascending or descending order).

ls -lS

9.16 Consider the following short Bourne shell script.

```
#!/bin/sh
echo –n "what day is it today? "
read x y z
echo "$x ^ ^ ^ $y ^ ^ ^ z"
```

What is the output of the script when the user enters `Today is Friday before a long weekend`