**STRING SORTING**

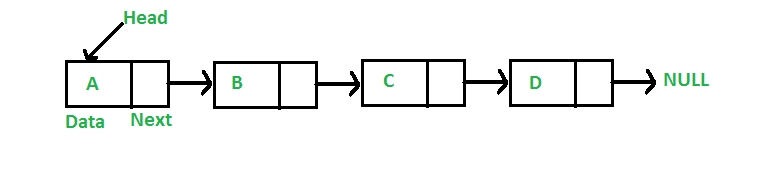
**(TO DISPLAY A SORTED LIST IN ALPHABETICAL ORDER)**

Logics used in string sorting:-

1. Linked lists
2. Pointers
3. Sorting

Linked lists:-

A linked list is a linear data structure, in which the elements are not stored at contiguous memory locations. The elements in a linked list are linked using pointers as shown in the below image:



In straightforward words, a coupled list consists of nodes wherever every node contains an information field and a reference(link) to ensuing node within the list.

Each component (we can decision it a node) of an inventory is comprising of 2 things - the info and a relevance ensuing node.

The last node has a reference to null.

The entry purpose into a coupled list is named the top of the list.

It ought to be noted that head isn't a separate node, however the relevance the primary node.

If the list is empty then the top may be a null reference.

A linked list is a dynamic data structure.

The number of nodes in a very list isn't mounted and might grow and shrink on demand.

Any application that has got to manage associate unknown variety of objects can have to be compelled to use a coupled list.

One disadvantage of a coupled list against associate array is that it doesn't enable direct access to the individual components.

If you would like to access a selected item then you've got to begin at the top and follow the references till you get to it item.

Another disadvantage is that a coupled list uses a lot of memory compare with associate array - we have a tendency to additional four bytes (on 32-bit CPU) to store a relevance ensuing node.

Declaring a Linked list :

In C language, a linked list can be implemented using structure and pointers .

struct LinkedList{

int data;

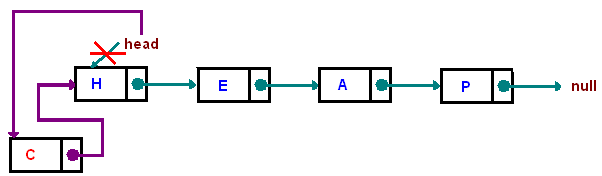
struct LinkedList \*next;

};

OPERATIONS ON LINKED LIST:

* addFirst

The method creates a node and prepends it at the beginning of the list.



public void addFirst(AnyType item)

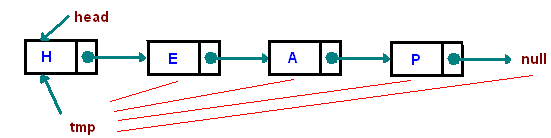
{

head = new Node<AnyType>(item, head);

}

* Traversing

Start with the head and access each node until you reach null. Do not change the head reference.

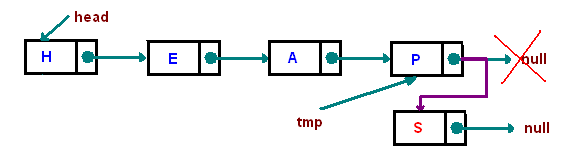


Node tmp = head;

while(tmp != null) tmp = tmp.next;

* addLast

The method appends the node to the end of the list. This requires traversing, but make sure you stop at the last node



public void addLast(AnyType item)

{

if(head == null) addFirst(item);

else

{

Node<AnyType> tmp = head;

while(tmp.next != null) tmp = tmp.next;

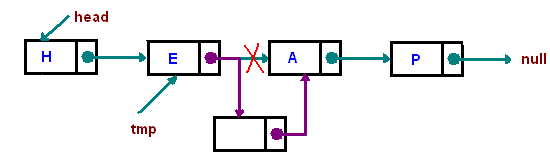
tmp.next = new Node<AnyType>(item, null);

}

}

* Inserting "after"

Find a node containing "key" and insert a new node after it. In the picture below, we insert a new node after "e":



public void insertAfter(AnyType key, AnyType toInsert)

{

Node<AnyType> tmp = head;

while(tmp != null && !tmp.data.equals(key)) tmp = tmp.next;

if(tmp != null)

tmp.next = new Node<AnyType>(toInsert, tmp.next);

}

* TYPE DEF:

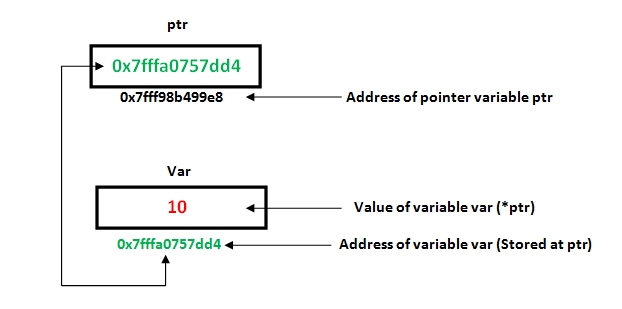
- The C programming language provides a keyword called typedef, which you can use to give a type a new name.

Pointers:-

Pointers may be a variable that stores/points the address of another variable.

A Pointer is used to allocate memory dynamically i.e. at run time.

The pointer variable may be happiness to any of the info kind like int, float, char, double, short etc.



* Pointer Syntax : data\_type \*var\_name; Example : int \*p;  char \*p;
* Where, \* is used to denote that “p” is pointer variable and not a normal variable.

Sorting:-

Sorting is nothing however transcription the info in ascending or falling order.

The term sorting came into image, as humans realized the importance of looking quickly.

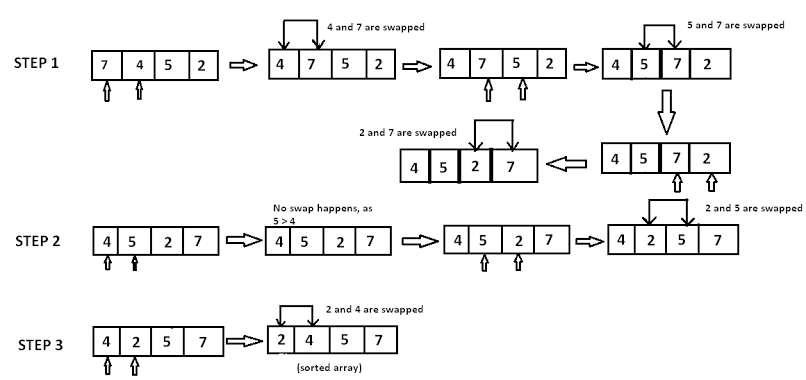
There area unit such a big amount of things in our reality that we'd like to go looking for, sort of a specific record in information, roll numbers in advantage list, a specific sign

in telephone book, a specific page in a very book etc.

All this may are a multitude if the info was unbroken unordered and unsorted, however luckily the idea of sorting came into existence, creating it easier for everybody

to arrange information in Associate in Nursing order, therefore creating it easier to go looking.

Sorting arranges information in a very sequence that makes looking easier.



* fgets is a function in the C programming language that reads a limited number of characters from a given file stream source into an array of characters.

NOTE:

* In this code we have also used functions:

| Function | Work of Function |
| --- | --- |
| [strlen()](https://www.programiz.com/c-programming/library-function/strlen) | computes string's length |
| [strcpy()](https://www.programiz.com/c-programming/library-function/strcpy) | copies a string to another |
| [strcat()](https://www.programiz.com/c-programming/library-function/strcat) | concatenates(joins) two strings |
| [strcmp()](https://www.programiz.com/c-programming/library-function/strcmp) | compares two strings |
| strlwr() | converts string to lowercase |
| strupr() | converts string to uppercase |

Algorithm:

1. Create a class Node which has two attributes: data and next. Next is a pointer to the next node in the list.
2. Create another class which has two attributes: head and tail.
3. addNode() will add a new node to the list:
   1. Create a new node.
   2. It first checks, whether the head is equal to null which means the list is empty.
   3. If the list is empty, both head and tail will point to a newly added node.
   4. If the list is not empty, the new node will be added to end of the list such that tail's next will point to a newly added node. This new node will become the new tail of the list.
4. A sorted list class will sort the nodes of the list in alphabetical order.
   1. Define a node current which will point to head.
   2. Define another node index which will point to node next to current.
   3. Compare data of current and index node. If current's data is greater than the index's data then, swap the data between them.
   4. Current will point to current.next and index will point to index.next.
   5. Continue this process until the entire list is sorted.
5. display() will display the nodes present in the list:
   1. Define a node current which will initially point to the head of the list.
   2. Traverse through the list till current points to null.
   3. Display each node by making current to point to node next to it in each iteration.

The code :-

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#define MAX\_STR\_LEN 20

//represents a node in a singly linked list

typedef struct link\_node {

char node\_str[MAX\_STR\_LEN];

struct link\_node \*next;

} node;

//declare the nodes and functions for sorting the list

int compare\_node(struct link\_node \*n1, struct link\_node \*n2);

struct link\_node \*add\_node(struct link\_node \*list, struct link\_node \*node);

void display\_list(struct link\_node \*head); //function to display list

void free\_list(node \*head); //function to free list

void cleanInput(char \*input); //function to clean input

int main() {

char input[MAX\_STR\_LEN];

node \*head, \*newNode; //declare nodes head and newNode

head = NULL;

printf("Enter a string\n");

do {

fgets(input, MAX\_STR\_LEN, stdin);// **f**gets stands for file get string.

cleanInput(input);

newNode = (node\*)malloc(sizeof(node)); //allocate memory using mallloc function

strcpy(newNode->node\_str, input);

newNode->next = NULL; //make next point NULL

if (input[0] != '\0') {

head = add\_node(head, newNode);

}

} while (input[0] != '\0'); //tranverse the list until input is the last node

display\_list(head);

free\_list(head);

return 0;

}

struct link\_node \*add\_node(struct link\_node \*list, struct link\_node \*node){

struct link\_node \*prev, \*next;

if (!list) {

list = node;

}

else {

prev = NULL;

next = list;

while (next && compare\_node(node,next)>0) {

prev = next;

next = next->next;

}

if (!next) {

prev->next = node;

}

else {

if (prev) {

node->next = prev->next;

prev->next = node;

}

else {

node->next = list;

list = node;

}

}

}

return list;

}

void free\_list(node \*head) {

node \*prev = head;

node \*cur = head;

while (cur) {

prev = cur;

cur = prev->next;

free(prev);

}

}

//code to compare the two nodes

int compare\_node(struct link\_node \*n1, struct link\_node \*n2) {

return strcmp(n1->node\_str, n2->node\_str);

}

void cleanInput(char\* input) {

int len = strlen(input);

int i;

for (i = 0; i < len-1; i++) {}

input[i] = '\0';

}

//display() will display all the nodes present in the list

void display\_list(struct link\_node \*head) {

while (head) {

printf("%s \n", head->node\_str);

head = head->next; //points the head node to the next node

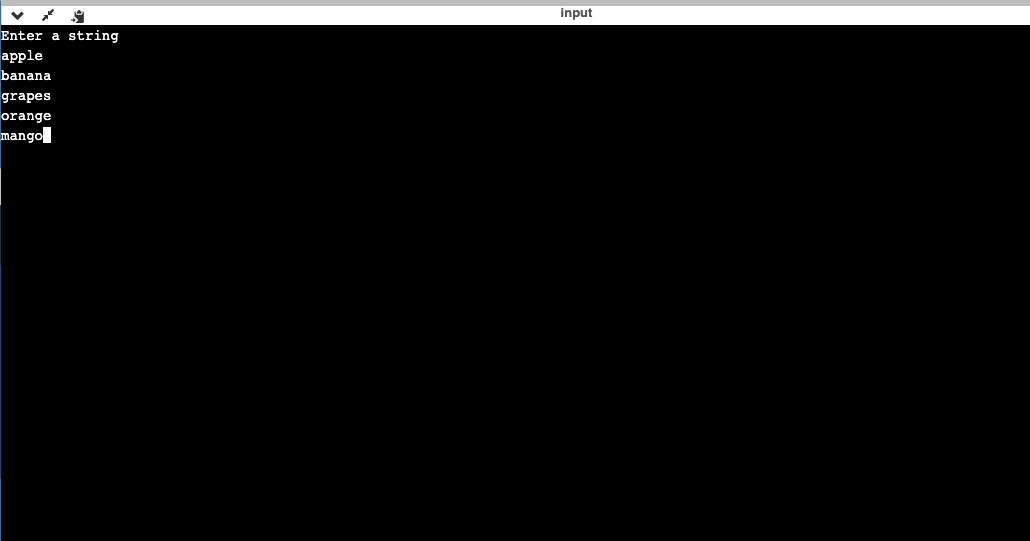
}

}

OUTPUT:

The Output will show the linked list in alphabetical order

1. Give an input of names:



1. Press enter twice and the list of names will be printed in alphabetical order



CONCLUDSION:

As you can see in the Output screenshots the list given in the program is sorted in alphabetical order.