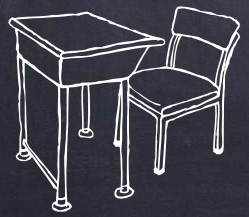
IT MINI PROJECT

Presented by
PARSAPU KAMAL RAJ



TOPIC

LEKON - HOSTEL MANAGEMENT

Purpose and Application

Hostel management is a real life problem like when we come to the college with our friends I am a hosteller I want a room to stay in the college so I go to the hostel office for room allotment there allocating the rooms to the students there asking the details and there allocating room number wise sequentially.

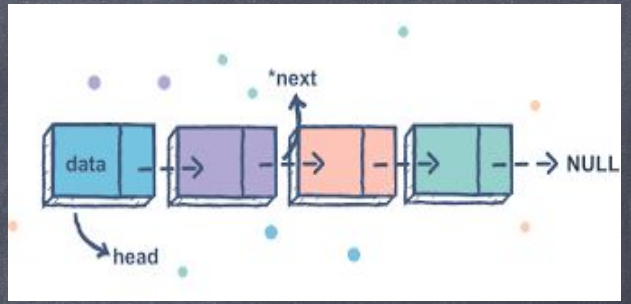
The main idea of this project is that we wanted to make the application of hostel management easy and simplier using the methods which we learnt in this course.

DATA STRUCTURE USED IN THIS PROJECT

*The data structure which we used was **SINGLY LINKED LIST** using C language.*

We are taking all the functionalities of the hostel office as the functions in our program.

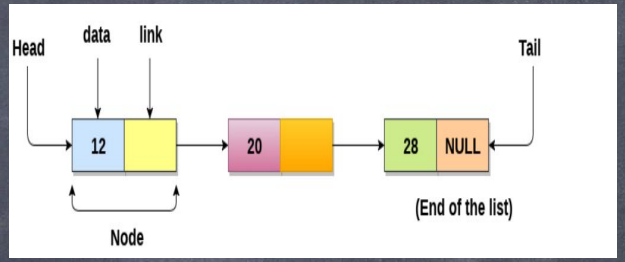
For allocating the rooms for the students we are taking each node as the one room we are filling the details of the student in the nodes data and inserting another node for the new room allotment in the sequentially by using traverse we can go to each node so we can go through each student details respective rooms.



What is Singly Linked List ??

Like arrays, Linked List is a linear data structure. Unlike arrays, linked list elements are not stored at a contiguous location; the elements are linked using pointers.

Linked List can be defined as collection of objects called nodes that are randomly stored in the memory.



A node contains two fields i.e. data stored at that particular address and the pointer which contains the address of the next node in the memory.
The last node of the list contains pointer to the null.

Singly linked list can be defined as the collection of ordered set of elements. The number of elements may vary according to need of the program.

A node in the singly linked list consist of two parts: data part and link part.

Data part of the node stores actual information that is to be represented by the node while the link part of the node stores the address of its immediate successor.

Justification for why using this Data structure

Here we use the singly linked list for creating the hostel management system for justifying this is the best data structure first we want to know the key difference between the singly linked list , stack and queue.

Stack is a linked list that allows insertion / removal only from its tail
queue is a linked list that allows insertion only at its tail and removal only from its head.

With the basic knowledge of operations on Linked Lists like insertion, search, traversing of elements in the Linked list, the hostel management and room allotment system can be created.

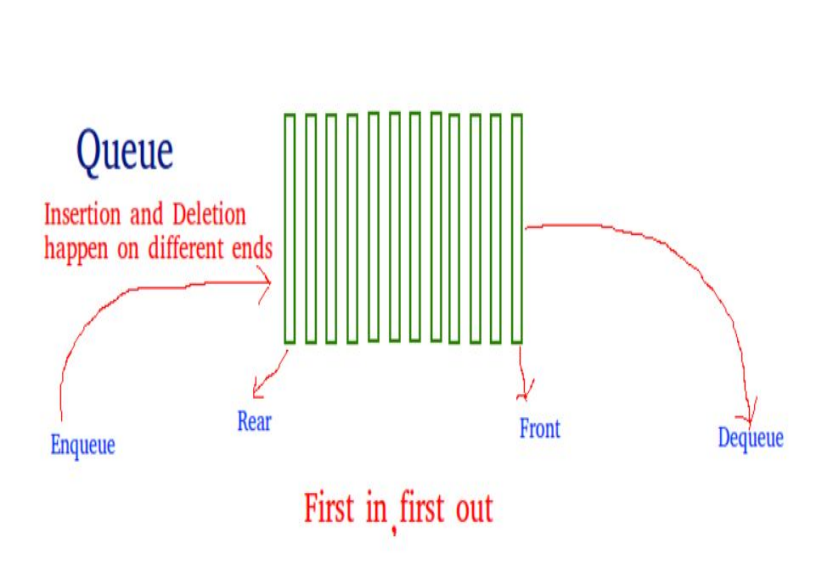
Stacks are based on the LIFO principle, i.e., the element inserted at the last, is the first element to come out of the list

Queues are based on the FIFO principle, i.e., the element inserted at the first, is the first element to come out of the list.

But in the singly Linked list we can insert the element in the middle of the linked list.

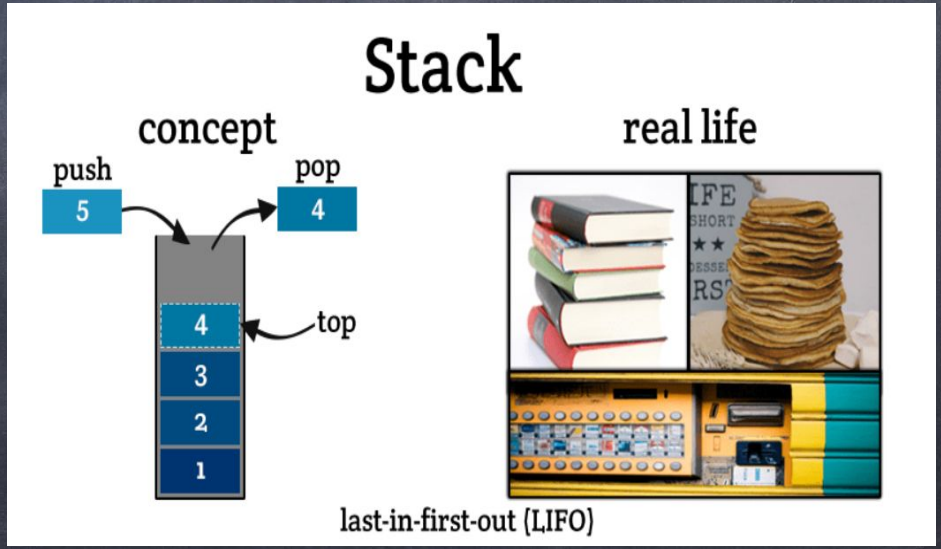
This is the key reason for selecting the singly linked list as our primary data structure because by using the stack or queue we cannot update or vacate the rooms in the middle hostel rooms .

We can only able to change the front and rear rooms means 1st or last rooms by using the stack or queue.



First In First Out (FIFO)

Last In First Out (LIFO)

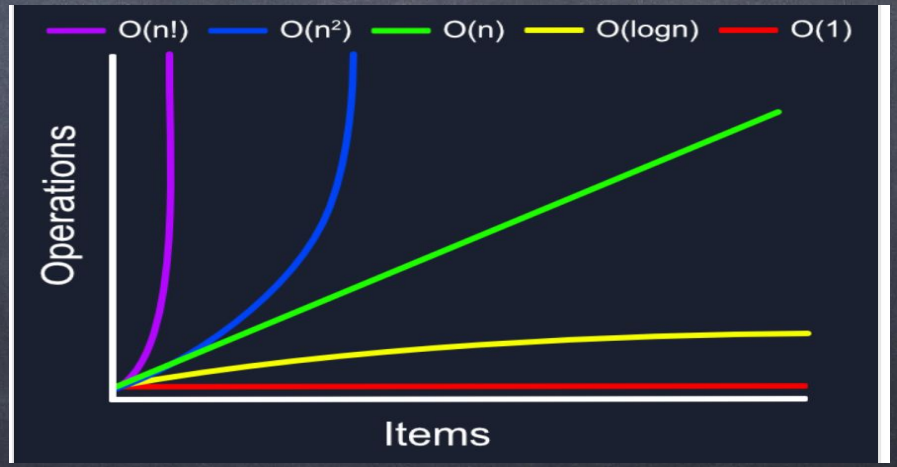


Time Complexity Analysis

The worst case Time Complexity for retrieving a node from anywhere in the list is O(n).

You can add a node at the front, the end or anywhere in the linked list.
The worst case Time Complexity for performing these operations is as follows:

- *Add item to the front of the list: O(1)*
- *Add item to the end of the list: O(n)*
- *Add item to anywhere in the list: O(n)*



Time Complexity for Linked Operations

Time Complexity for the Linked Data Structure		
Operation	**Cost**	
	Singly Linked	**Doubly Linked**
read (anywhere in the linked list)	O(n)	O(n)
add/remove (at the head)	O(1)	O(1)
add/remove (at the tail)	O(n)	O(1)
add/remove (in the interior of the structure)	O(n)	O(n)
resize	N/A	N/A
find by position	O(n)	O(n)
find by target	O(n)	O(n)

You can remove a node either from the front, the end or from anywhere in the list. The worst case Time Complexity for performing this operation is as follows:

- **Remove item from the front of the list: O(1)**
- **Remove item from the end of the list: O(n)**
- **Remove item from anywhere in the list: O(n)**

Conclusion

We can allocate n rooms for n students in college.In our program we not delete any node we just vacate the node by replacing previous data with NULL why??

Because deleting the node means deleting the room it's not possible just we want to erase the previous room data and make it ready for new allotment if we want to delete the node we again change the next room person to the before room like we want shift till the last in real world this work become complex.

And not possible that's why we vacate the room means erasing the details put it for the new allotment so other node means no other room students not disturbed this is the thing we observe in our program , with our program we did maximum functionalities like hostel office software.

Finally, we concluded that we want to update our program by implementing the some other functionalities like issues in hostel rooms , resolving them,mess allocation , fee payments and feedback from the students.

THANK YOU