**DATA STRUCTURES**

**AND**

**APPLICATIONS**

***MINI PROJECT***

**TODO LIST**

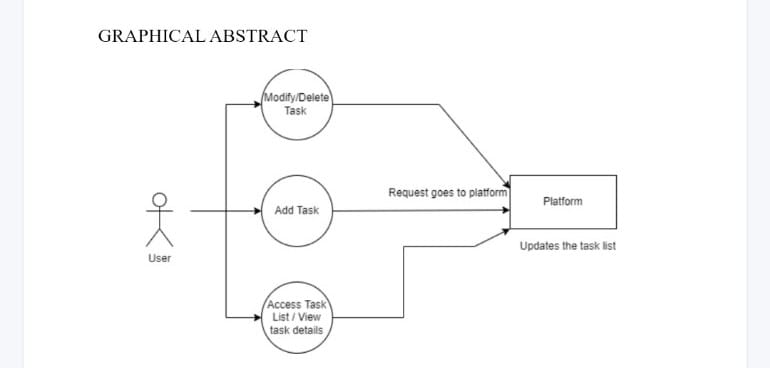
***SRIDHAR E -****71772118145*

**ABSTRACT**

A to-do list is a list of tasks that need to be completed, typically organized in order of priority. It is one of the simplest solutions for task management and provides a minimal and elegant way for managing tasks a person wishes to accomplish.

Our aim is to design a simple and elegant program for people to keep a track of the status of their tasks. Making a to-do list is an easy and important task that everyone should do. The immense satisfaction that one gets when completing the task and marking it on the list are incomparable. Moreover, creating a list of tasks ensure you don't miss out on anything. It's a scientific fact that when you write the tasks that you need to complete, you are even more motivated to complete it. With this in mind, we come to build a platform which will help people create their own task list.

With the help of modern tools and technologies, we strive to build a minimal and efficient to-do list which minimizes distractions and helps people achieve task management with ease and without hassle.

******

**INTRODUCTION**

A to-do list is a simple prioritized list of the tasks a person must complete. People make a list of everything they need to do, ranked according to priority from the most critical task at the top to the least critical task at the bottom

A few of the features of a good to-do list application include:

1)Plan and execute simple actions.

2)Prioritize, manage, and reason about tasks.

3)Record notes, action items and ideas

To-dos are the tasks or the atomic entities that make up a to-do list. To-dos are made quickly, the bulk of them do not specify the work; instead, they are typically just comprehensive enough to serve as a valuable indicator. To be sure, to-do terminology like "Groceries" or "Car Wash" is frequently grammatically correct. Because the signal is so quick, it is only useful for a short period of time while the task is remembered. In certain cases, a simple item like a stack is enough to recall the job without the need for a note.

There are clear immediate implications to adding a to-do list to a person's productivity system. The functionalities provided by a good to-do list application/system help declutter the user's mind as their pending tasks are recorded safely and they won't be forgotten.

The To-do list project is a user-friendly website which helps them to keep a track of their tasks. It is a simple site which requires no sign-in log-in or any personal details but still records your task, mark the completed tasks, and stores them even if you visit the site after a few days.

Traditionally, they're written on a piece of paper or post it notes and acts as a memory aid. As technology has evolved, we have been able to create a to-do lists with excel spreadsheets, word documents, to-do list apps, Microsoft to do and google to do list to name a few. You can use to do list in your home and personal life, or in the workplace.

**BENEFITS OF TODO LIST**

1. Improves your memory. A to do list acts as an external memory aid. It's only possible to hold a few pieces of information at one time. Keep a to do list and you'll be able to keep track of everything, rather than just a few of the tasks you need to do. Your to do list will also reinforce the information, which makes it less likely you're going to forget something.

2. Increases productivity. A to do list allows you to prioritize the tasks that are more important. This means you don't waste time on tasks that don't require your immediate attention. Your list will help you stay focused on the tasks that are the most important.

3. Helps with motivation: To do lists are a great motivational tool because you can use them to clarify your goals. You can divide your long-term goal into smaller, more achievable short-term goals and as you tick each one off your list, your confidence will increase

**EVOLUTION OF TODO LIST**

The idea of plan for the day has existed for quite a while and it is one of the essential techniques for the board of assignments, utilization of a tasks as an update framework, tasks as a framework for note the executives, and so on. In the least difficult and most crude structure, a plan for the day can be executed on a pen and paper as an agenda of things which can be crossed of or ticked against when finished.

This can be additionally reached out to schedules, by composing undertakings against dates where the dates can likewise go about as cutoff times for specific assignments Other potential augmentations of plans for the day can be on whiteboards, diaries, text editors.etc.

The functionalities of plans for the day normally develop to fit web applications and applications on advanced gadgets flawlessly. Furnished with current apparatuses and advancements, specialists can fabricate an application to make a negligible and strong application that can assist with supporting efficiency without loss of concentration and consideration.

With the processing power and steadiness of current gadgets and data sets, forgetting about assignments won't be an issue individuals should confront any longer and they can have confidence, just centered around the errands they should achieve similarly as with present day innovation and the force of advanced gadgets, combination will he consistent and undertakings can be adjusted across different gadgets at the same time, with next to no problem.

**DATA STRUCTURE**

The data structure used here is linked list.

Let us see about linked list in detail

**LINKED LIST**

A linked list is the most sought-after data structure when it comes to handling dynamic data elements. A linked list consists of a data element known as a node. And each node consists of two fields: one field has data, and in the second field, the node has an address that keeps a reference to the next node.

What is a Linked List?

A linked list is a linear data structure that stores a collection of data elements dynamically.

Nodes represent those data elements, and links or pointers connect each node.

Each node consists of two fields, the information stored in a linked list and a pointer that stores the address of its next node.

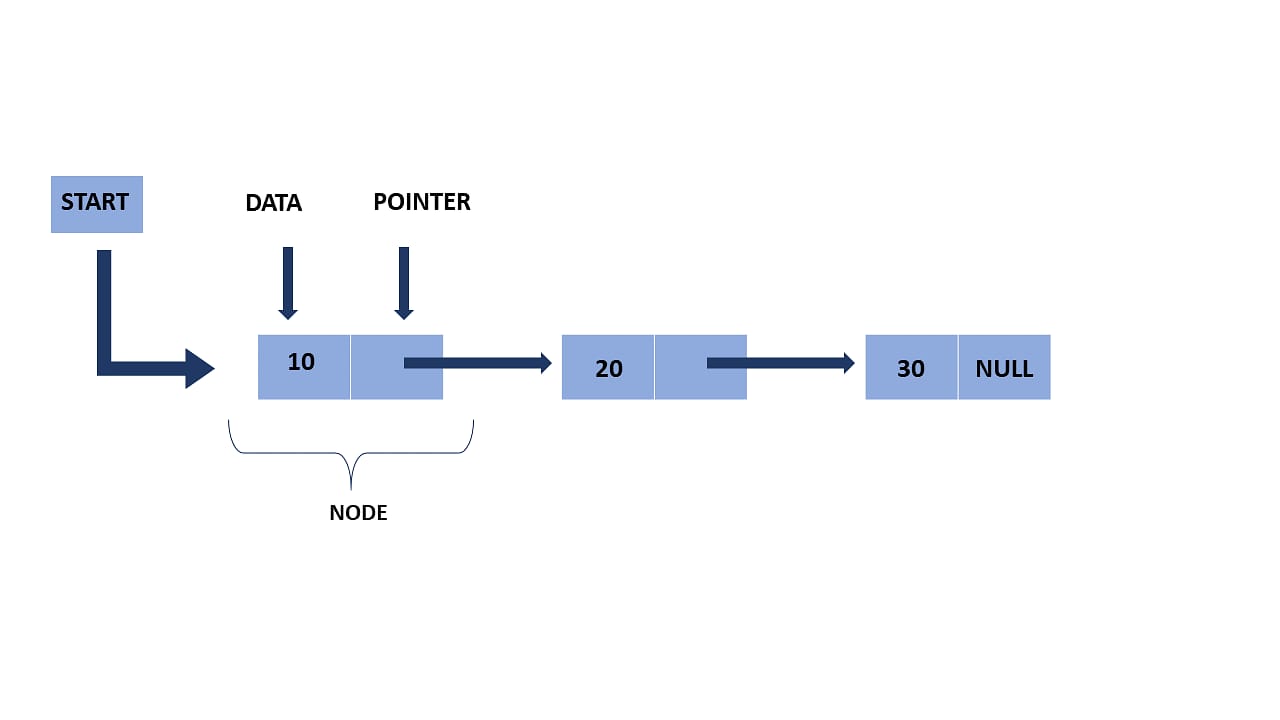
The last node contains null in its second field because it will point to no node.

A linked list can grow and shrink its size, as per the requirement.

It does not waste memory space.

**Representation of a Linked List**

This representation of a linked list depicts that each node consists of two fields. The first field consists of data, and the second field consists of pointers that point to another node.



Here, the start pointer stores the address of the first node, and at the end, there is a null pointer that states the end of the Linked List.

**Creation of Node and Declaration of Linked Lists**

struct node

{

int data;

struct node \* next;

};

struct node \* n;

n=(struct node\*)malloc(sizeof(struct node\*));

It is a declaration of a node that consists of the first variable as data and the next as a pointer, which will keep the address of the next node.

Here you need to use the malloc function to allocate memory for the nodes dynamically.

**Types of Linked Lists**

The linked list mainly has three types, they are:

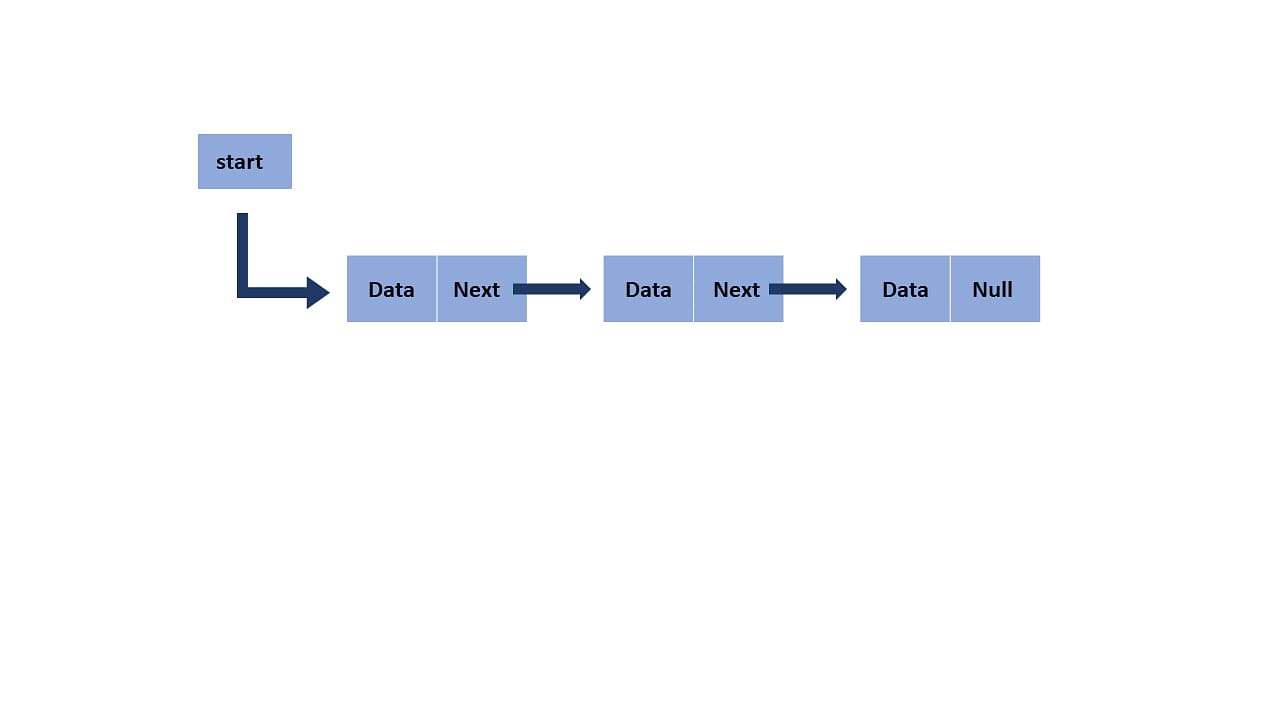
Singly Linked List

Doubly Linked List

Circular Linked List

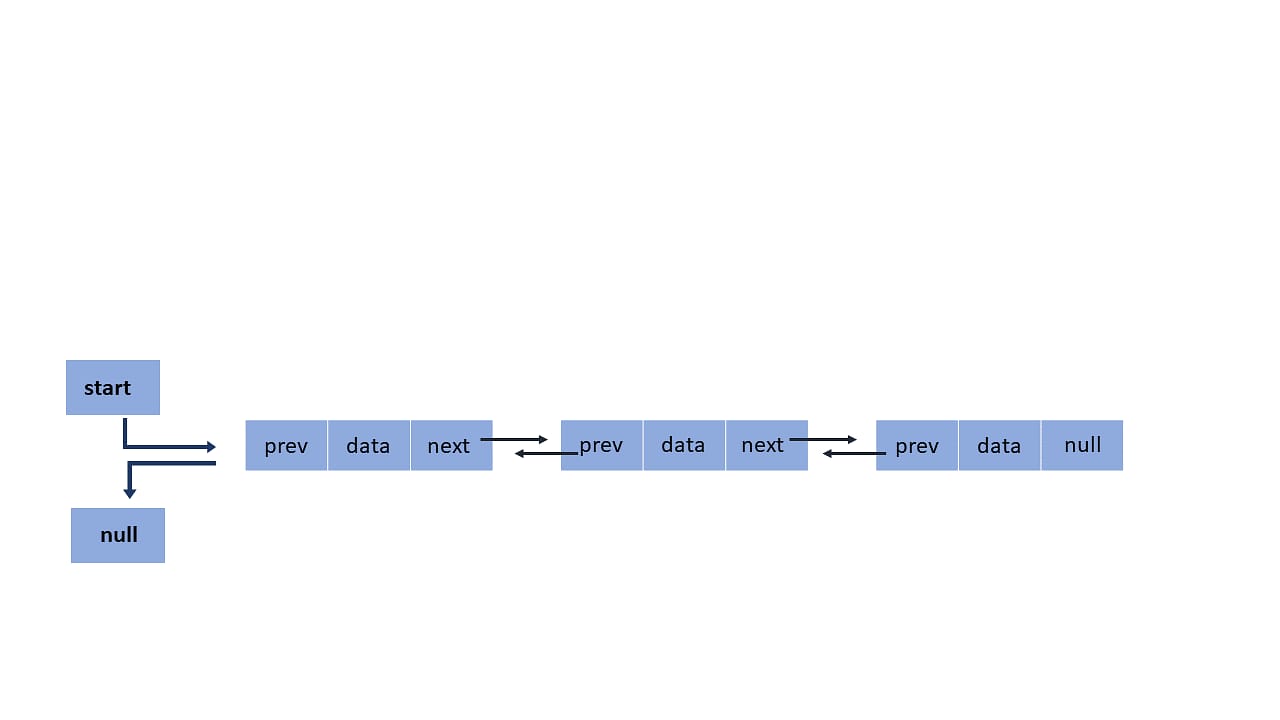
**Singly Linked List**

A singly linked list is the most common type of linked list. Each node has data and an address field that contains a reference to the next node



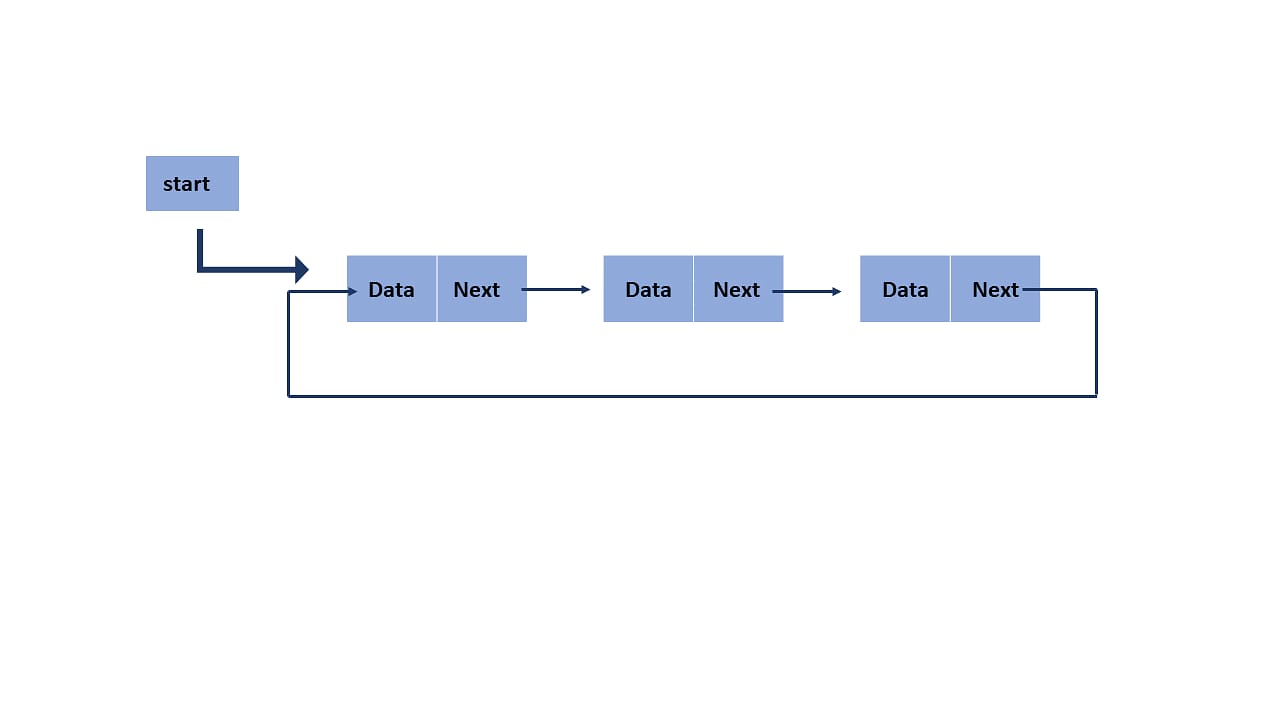
**Doubly Linked List**

There are two pointer storage blocks in the doubly linked list. The first pointer block in each node stores the address of the previous node. Hence, in the doubly linked inventory, there are three fields that are the previous pointers, that contain a reference to the previous node. Then there is the data, and last you have the next pointer, which points to the next node. Thus, you can go in both directions (backward and forward)



**Circular Linked List**

The circular linked list is extremely similar to the singly linked list. The only difference is that the last node is connected with the first node, forming a circular loop in the circular linked list.

 Circular link lists can either be singly or doubly-linked lists.The next node's next pointer will point to the first node to form a singly linked list.

The previous pointer of the first node keeps the address of the last node to form a doubly-linked list.

**Essential Operation on Linked Lists**

Traversing: To traverse all nodes one by one.

Insertion: To insert new nodes at specific positions.

Deletion: To delete nodes from specific positions.

Searching: To search for an element from the linked list.

Traversal

In this operation, you will display all the nodes in the linked list.

When the temp is null, it means you traversed all the nodes, and you reach the end of the linked list and get out from the while loop. In this operation, you will display all the nodes in the linked list.

struct node \* temp = start;

printf(“\n list empty are-”);

while (temp!= NULL)

{

printf(‘%d “, temp -> data)

temp=temp -> next;

}

Insertion

You can add a node at the beginning, middle, and end.

Insert at the Beginning

Create a memory for a new node.

Store data in a new node.

Change next to the new node to point to start.

Change starts to tell the recently created node.

struct node \*NewNode;

NewNode=malloc(sizeof(struct node));

NewNode -> data = 40;

NewNode -> next= start;

start= NewNode;

Insert at the End

Insert a new node and store data in it.

Traverse the last node of a linked list

Change the next pointer of the last node to the newly created node.

struct node \*NewNode;

NewNode=malloc(sizeof(struct node));

NewNode-> data = 40;

NewNode->next = NULL;

struct node \*temp = start;

while( temp->next ! = NULL){

temp=temp -> next;

}

temp -> next = NewNode;

Insert at the Middle

Allocate memory and store data in the new node.

Traverse the node, which is just before the new node.

Change the next pointer to add a new node in between.

struct node \*NewNode;

NewNode= malloc(sizeof(struct node));

NewNode -> data = 40;

struct node - > temp = start;

for(int i=2; i<position; i++){

if (temp -> next!= NULL)

temp = temp -> next;

}}

NewNode -> next = temp -> next;

temp -> next = NewNode;

Deletion

You can also do deletion in the linked list in three ways either from the end, beginning, or from a specific position.

Delete from the Beginning

The point starts at the second node.

start = start -> next;

Delete from the End

Traverse the second last element in the linked list.

Change its next pointer to null.

struct node \* temp = start;

while(temp -> next -> next!= NULL){

temp=temp -> next;

}

temp -> next = NULL;

Delete from the Middle

Traverse the element before the element to be deleted.

Change the next pointer to exclude the node from the linked list.

for (int i = 2; i, position; i++){

if (temp -> next ! = NULL)

temp = temp -> next;

}}

temp-> next = temp -> next -> next;

Searching

The search operation is done to find a particular element in the linked list. If the element is found in any location, then it returns. Else, it will return null.

**Application of a Linked List**

A linked list is used to implement stacks and queues.

A linked list also helps to implement an adjacency matrix graph.

It is used for the dynamic memory location.

The linked list makes it easy to deal with the addition and multiplication of polynomial operations.

Implementing a hash table, each bucket of the hash table itself behaves as a linked list.

It is used in a functionality known as undo in Photoshop and Word.

**PROGRAM FOR TODO LIST**

**#include <stdio.h>**

**#include <stdlib.h>**

**#include<string.h>**

**int found=0,test=0;**

**typedef struct ToDo todo;**

**void sort();**

**struct ToDo**

**{**

**char data[100];**

**int count;**

**int date;**

**int month;**

**int year;**

**todo \*link;**

**};**

**int datevalid(int dd,int mm,int yy)**

**{**

**if(yy>=2022)**

**{**

**if(mm>=1 && mm<=12)**

**{**

**if((dd>=1 && dd<=31) && (mm==1 || mm==3 || mm==5 || mm==7 || mm==8 || mm==10 || mm==12))**

**return 1;**

**else if((dd>=1 && dd<=30) && (mm==4 || mm==6 || mm==9 || mm==11))**

**return 1;**

**else if((dd>=1 && dd<=28) && (mm==2))**

**return 1;**

**else if(dd==29 && mm==2 && (yy%400==0 ||(yy%4==0 && yy%100!=0)))**

**return 1;**

**else**

**return 0;**

**}**

**else**

**{**

**return 0;**

**}**

**}**

**else**

**{**

**return 0;**

**}**

**}**

**void fixcount();**

**todo \*start=NULL;**

**void welcomeUser()**

**{**

**system("color 4F");**

**printf("\n\n\n\n\n");**

**printf("\t------------------------------------------------------------------------------------------------------\n\n");**

**printf("\t#################################### TODO LIST APP ##############################################\n\n");**

**printf("\t------------------------------------------------------------------------------------------------------");**

**printf("\n\n\n\t\t\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*WELCOME\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n\n\n\n\n\n\n\n\n\t");**

**system("pause");**

**}**

**void seeToDo()**

**{ if(test==0)**

**{**

**system("cls");**

**}**

**todo \*temp;**

**temp=start;**

**printf("YOUR TODO LIST:\n");**

**if(start==NULL)**

**{**

**printf("\n");**

**printf("\nEmpty TODO\n\n");**

**}**

**while(temp!=NULL)**

**{**

**printf("%d)",temp->count);**

**puts(temp->data);**

**fflush(stdin);**

**printf("\tDeadline:");**

**printf("%d/%d/%d",temp->date,temp->month,temp->year);**

**temp=temp->link;**

**printf("\n");**

**}**

**printf("\n\n\n");**

**system("pause");**

**}**

**void createToDo()**

**{**

**char k;**

**char choice;**

**char date[50];**

**todo \*t,\*temp;**

**t=NULL;**

**do**

**{**

**printf("\n");**

**printf("\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*ADD YOUR TASK\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");**

**printf("\n");**

**if(start==NULL)**

**{**

**t=(todo \*)calloc(1,sizeof(todo));**

**start=t;**

**printf("\*");**

**fflush(stdin);**

**printf("Enter Task:");**

**gets(t->data);**

**printf("DUE DATE:");**

**fflush(stdin);**

**// gets(date);**

**// scanf("%d%d%d",&t->date,&t->month,&t->year);**

**printf("\nenter date:");**

**scanf("%d",&t->date);**

**printf("\nenter month:");**

**scanf("%d",&t->month);**

**printf("\nenter year:");**

**scanf("%d",&t->year);**

**if(datevalid(t->date,t->month,t->year))**

**{**

**printf("\n");**

**}**

**else**

**{**

**printf("Invalid date or month or year ");**

**free(t);**

**start=NULL;**

**createToDo();**

**found=1;**

**return;**

**// t=(todo\*)malloc(sizeof(todo));**

**}**

**t->count=1;**

**start->link=NULL;**

**}**

**else**

**{**

**t=start;**

**temp=(todo \*)calloc(1,sizeof(todo));**

**printf("\*");**

**fflush(stdin);**

**gets(temp->data);**

**printf("DUE DATE:");**

**fflush(stdin);**

**printf("\nenter date:");**

**scanf("%d",&temp->date);**

**printf("\nenter month:");**

**scanf("%d",&temp->month);**

**printf("\nenter year:");**

**scanf("%d",&temp->year);**

**if( datevalid(temp->date,temp->month,temp->year))**

**{**

**printf("\n");**

**}**

**else**

**{**

**printf("Invalid date or month or year ");**

**free(temp);**

**createToDo();**

**return;**

**}**

**temp->link=NULL;**

**while(t->link!=NULL)**

**{**

**t=t->link;**

**}**

**t->link=temp;**

**}**

**sort();**

**fixcount();**

**printf("Task Added successfully");**

**printf("\n");**

**printf("\n");**

**printf("PRESS Y TO ADD NEW TASK:");**

**scanf("\n");**

**scanf("%c",&choice);**

**}**

**while(choice=='Y');**

**}**

**void delToDo()**

**{ if(test==0)**

**{**

**system("cls");**

**}**

**int d;**

**todo \*temp1,\*temp;**

**int i=0,n;**

**seeToDo();**

**printf("\nEnter the no of the Todo you want to remove:\n");**

**scanf("\n");**

**scanf("%d",&d);**

**temp1=start;**

**while(temp1!=NULL)**

**{ i++;**

**temp1=temp1->link;**

**}**

**if(i<d||d<0)**

**{**

**printf("invalid choice\n");**

**test=1;**

**free(temp1);**

**delToDo();**

**return;**

**}**

**//fixcount();**

**temp1=start;**

**temp=start->link;**

**while(1)**

**{**

**if(temp1->count==d)**

**{**

**start=start->link;**

**free(temp1);**

**fixcount();**

**break;**

**}**

**else if(temp->count==d)**

**{**

**temp1->link=temp->link;**

**free(temp);**

**fixcount();**

**break;**

**}**

**else**

**{**

**temp1=temp;**

**temp=temp->link;**

**}**

**}**

**system("pause");**

**}**

**void fixcount()**

**{**

**todo \*temp;**

**int i=1;**

**temp=start;**

**while(temp!=NULL)**

**{**

**temp->count=i;**

**i++;**

**temp=temp->link;**

**}**

**}**

**void updateToDo()**

**{**

**if(found==0)**

**{**

**system("cls");**

**}**

**todo \*temp,\*t;**

**char k;**

**printf("\nPress Y TO Add new todo :");**

**fflush(stdin);**

**scanf("%c",&k);**

**if(k=='Y')**

**{**

**temp=(todo \*)calloc(1,sizeof(todo));**

**fflush(stdin);**

**printf("ADD NEW TASK...");**

**printf("\n");**

**gets(temp->data);**

**printf("DUE DATE:");**

**printf("\nenter date:");**

**scanf("%d",&temp->date);**

**printf("\nenter month:");**

**scanf("%d",&temp->month);**

**printf("\nenter year:");**

**scanf("%d",&temp->year);**

**if( datevalid(temp->date,temp->month,temp->year))**

**{**

**printf("\n");**

**}**

**else**

**{**

**printf("Invalid date or month or year ");**

**found=1;**

**free(temp);**

**updateToDo();**

**return;**

**//goto h;**

**}**

**temp->link=NULL;**

**t=start;**

**while(t->link!=NULL)**

**{**

**t=t->link;**

**}**

**t->link=temp;**

**sort();**

**fixcount();**

**printf("Task Added successfully...");**

**}**

**printf("\n\n");**

**system("pause");**

**}**

**void sort()**

**{**

**todo \*temp1,\*temp=NULL;**

**temp1=start;**

**while(temp1!=NULL)**

**{**

**temp=temp1->link;**

**while(temp!=NULL)**

**{**

**if(temp1->year>temp->year)**

**{**

**swap(temp1,temp);**

**}**

**else if((temp1->year==temp->year)&&(temp1->month>temp->month))**

**{**

**swap(temp1,temp);**

**}**

**else if((temp1->year==temp->year)&&(temp1->month==temp->month)&&(temp1->date>temp->date))**

**{**

**swap(temp1,temp);**

**}**

**temp=temp->link;**

**}**

**temp1=temp1->link;**

**}**

**}**

**void swap(todo \*temp1,todo \*temp)**

**{**

**int a,b,c;**

**char word[100];**

**todo \*t;**

**strcpy(word,temp1->data);**

**strcpy(temp1->data,temp->data);**

**strcpy(temp->data,word);**

**a=temp1->year;**

**temp1->year=temp->year;**

**temp->year=a;**

**b=temp1->date;**

**temp1->date=temp->date;**

**temp->date=b;**

**c=temp1->month;**

**temp1->month=temp->month;**

**temp->month=c;**

**}**

**int main()**

**{**

**int choice,clear=0;**

**char opt;**

**welcomeUser();**

**system("cls");**

**createToDo();**

**while(1)**

**{**

**system("color 2F");**

**if(clear==0)**

**{**

**system("cls");**

**}**

**printf("\n>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>MENU:<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<\n");**

**printf("\n1.See Your ToDo List");**

**printf("\n2.Delete Your ToDos");**

**printf("\n3.Update Your ToDos");**

**printf("\n4.Exit");**

**printf("\n\nEnter your choice...............\n");**

**scanf("\n");**

**scanf("%d",&choice);**

**switch(choice)**

**{**

**case 1:**

**seeToDo();**

**break;**

**case 2:**

**delToDo();**

**break;**

**case 3:**

**system("cls");**

**updateToDo();**

**break;**

**case 4:**

**exit(0);**

**default:**

**printf("Invalid choice!!!");**

**clear=1;**

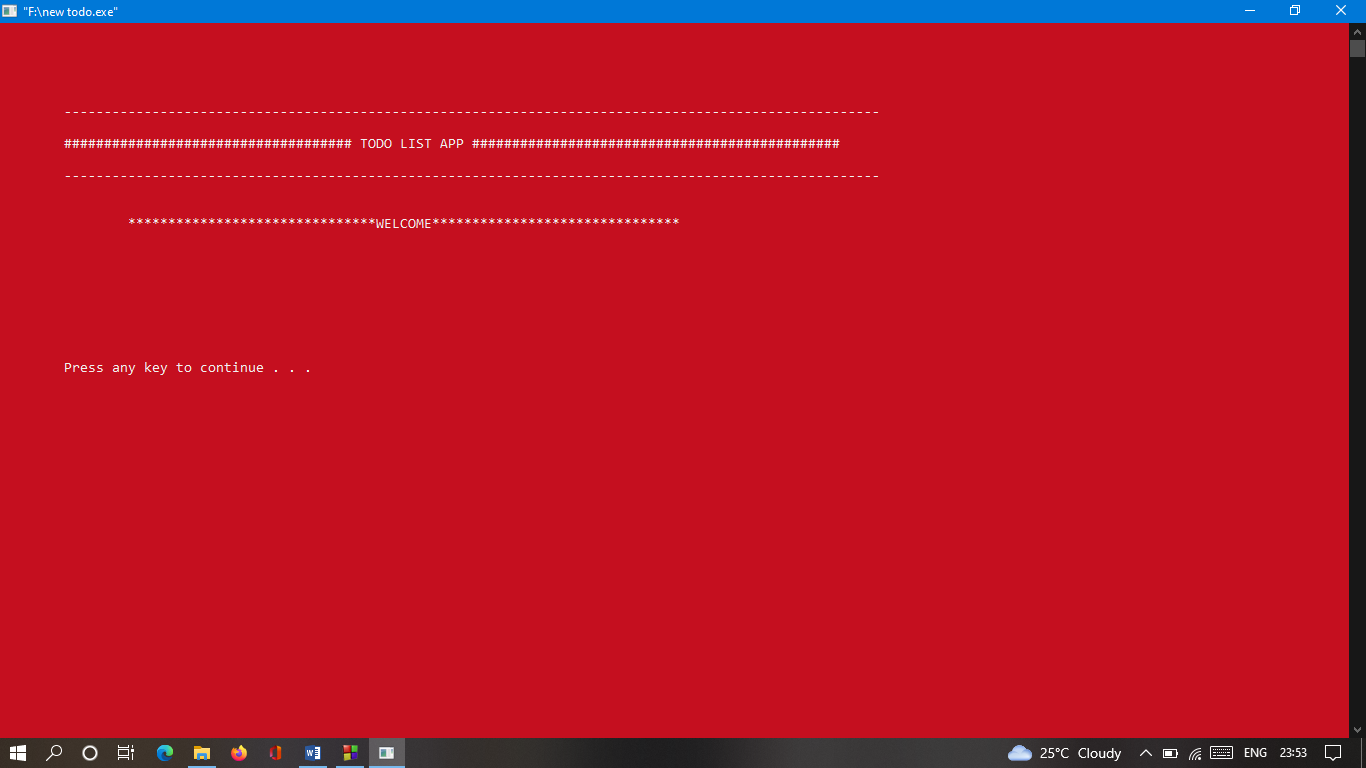
**}**

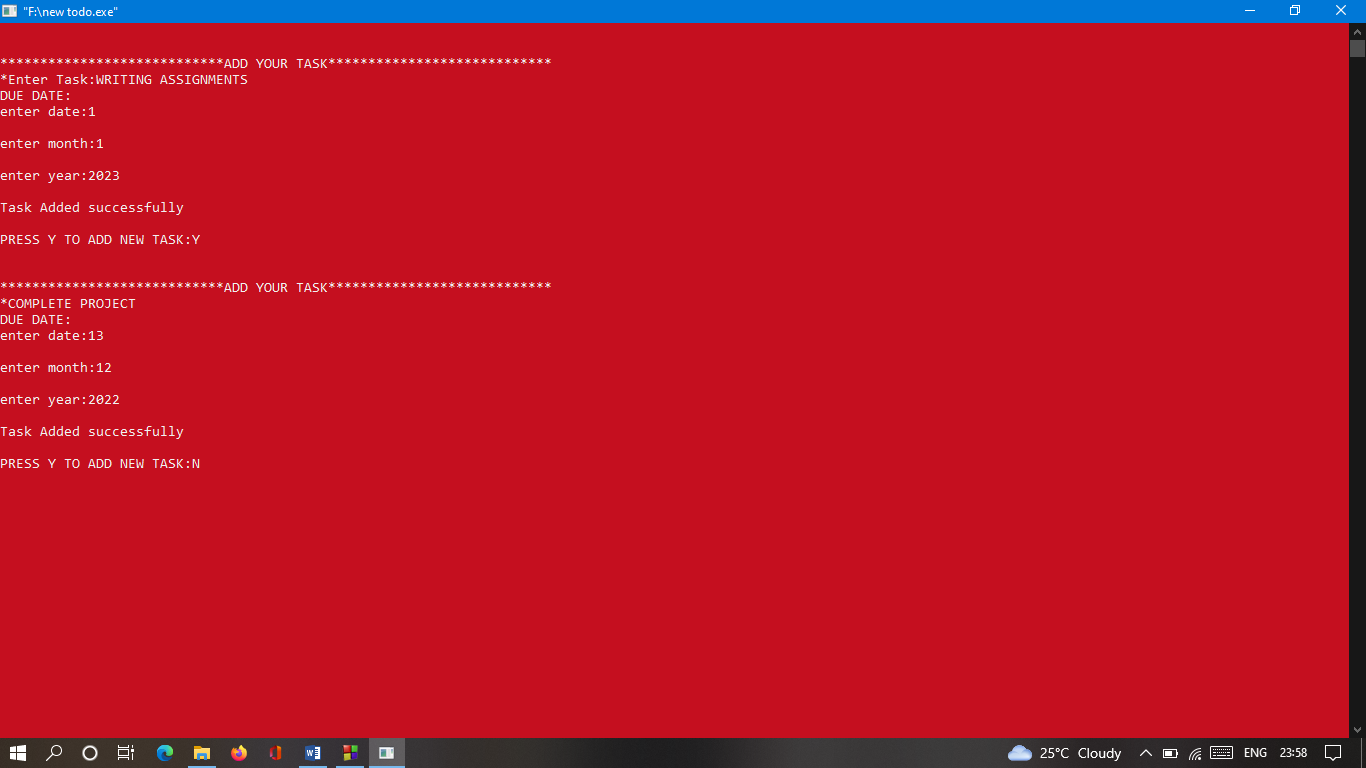
**} return 0;**

**}**

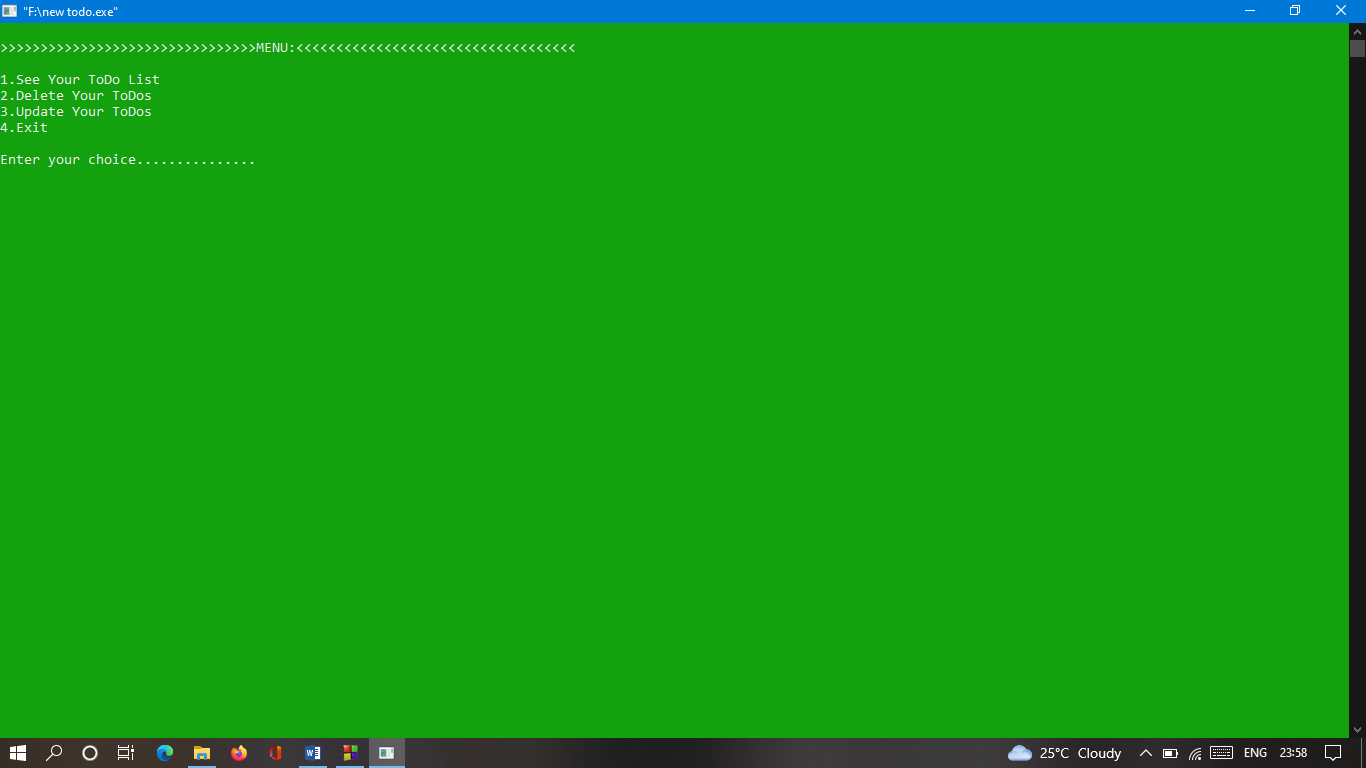
**OUTPUT**

ADDING TASKS

******

****

DISPLAYING MENU

****

SEEING TODO LIST TASKS

****

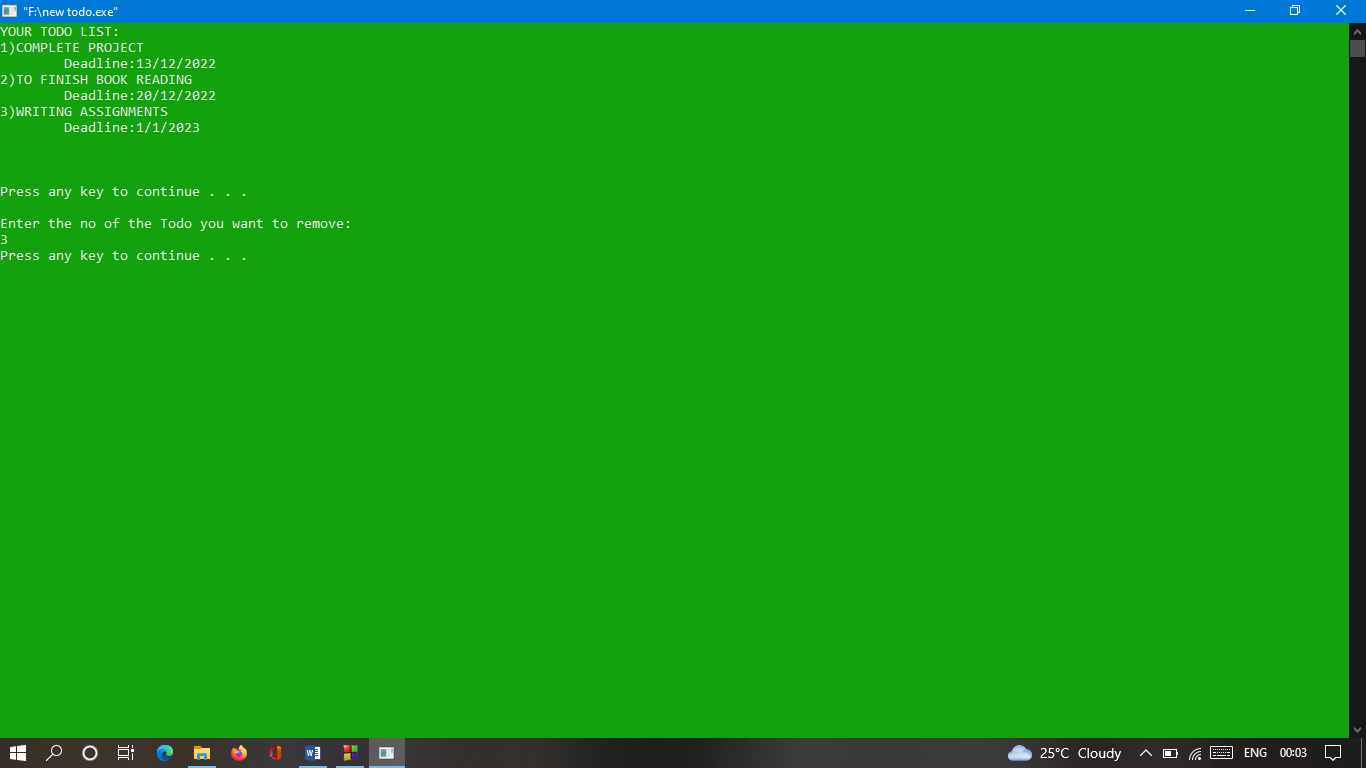
UPDATING TODO LIST TASKS

****

AFTER UPDATE,SEEING TODO LIST TASKS

****

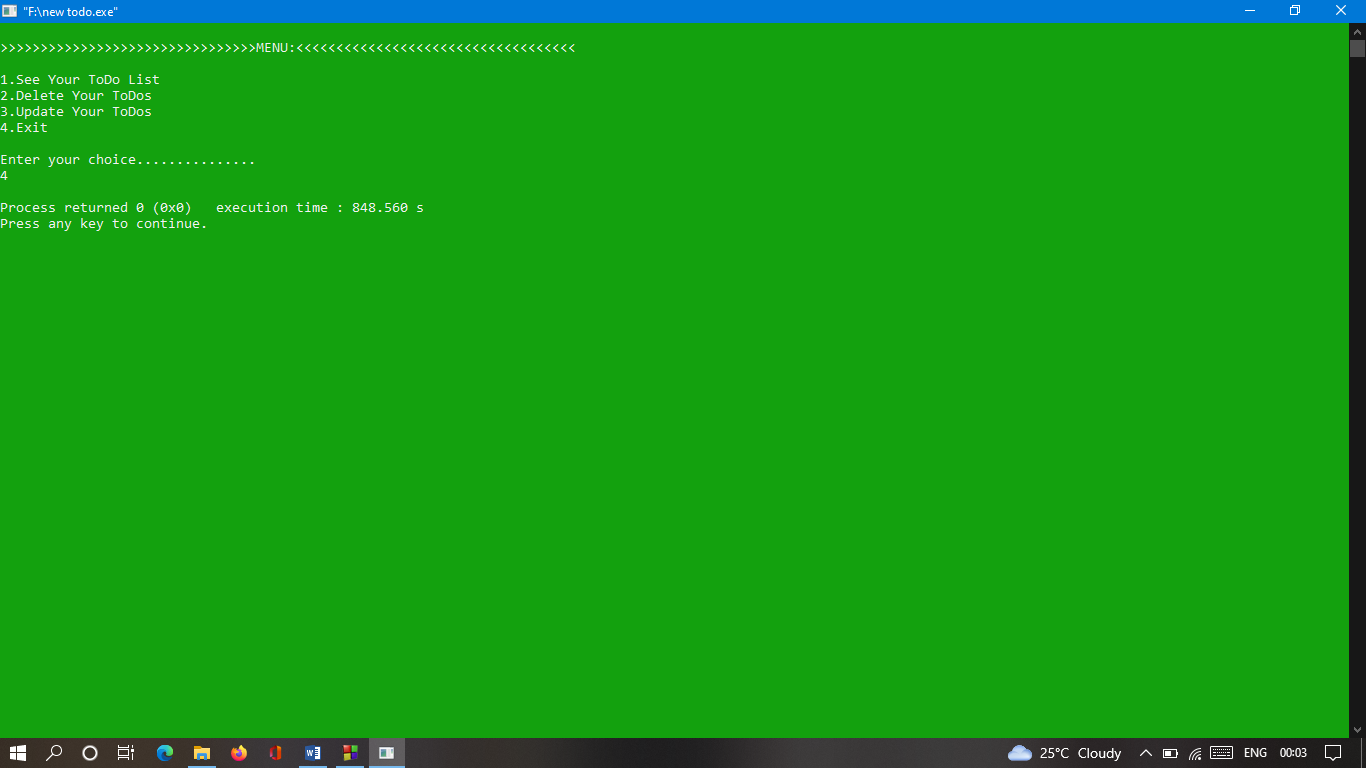
DELETING AFTER COMPLETING TASKS

****

DISPLAYING AFTER DELETING TASKS

****

EXITING TODO LIST

****

**SCOPE OF TODO LIST**

To start with, it is better to mention that a typical to-do list (also called as a list of things, a list of errands, things-to-do list, etc.) is just a list of tasks for a certain period of time, a day or a week.

Initially, to-do list was created for the purpose to be a well-structured work plan helps every person to see his current load and execute all the necessary tasks. Now it becomes a common practice almost for everyone, and it solves many problems concerning the high workload.

To-do list has its following functions:

Organization of a proper work with an accurate planning of every task with its short description

Simplify the workflow and a timely execution of necessary work scope

Improvement of every employee’s productivity and time management skills.

**THANK YOU**