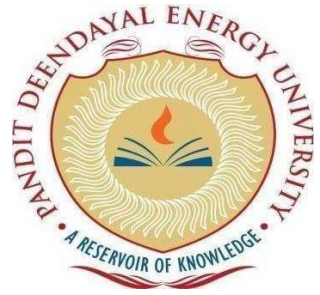
Pandit Deendayal Energy University, Gandhinagar
School of Technology
Department of Computer Science and Engineering



DATA STRUCTURES LAB
(20CP201P)

Course Coordinator:

Dr. Rutvij H. Jhaveri

Team:

JAIVIK JARIWALA 21BCP004
AVANI CHATURVEDI 21BCP019
FELIX NSHUTI 21BCP027
PARTH MISTRY 21BCP028
NITI CHOVATIA 21BCP032
ADITI PATHAK 21BCP428D

Semester: 3

Division: 1 **Group:** G1

INDEX

SR NO	NAME OF THE PROJECT	SIGNATURE
1	VENDICINE (Basic structures)	
2	DATABASE MANAGEMENT FILE READER AND GENERATOR (Stack)	
3	RESTAURANT MANAGEMENT (Queue)	
4	MUSIC PLAYER (Linked list)	
5	STORE MANAGEMENT (Trees)	
6	GRAPHICAL REPRSENTATION OF SHORTEST PATH PROBLEM (Graphs)	

PROJECT 1 – VENDICINE: E-VENDING MACHINE

DESCRIPTION:

The simulation of a vending Machine in C language it covers the inventory tracking and sales operation between seller and the buyer. The project is designed using C structures to enable all the operations and has two main parts which are the Seller login option and the buyer login option.

There are 3 header files namely client.h, seller.h, inventory.h and one main file named as main.c

CODE:

client.h

```
/*
The Client side of the application it is holding two main functions
The get_item() function which will help to get item from our inventory based on input item code.
The select_item() function for helping the client to select item to buy from the stock.
*/
#include <stdio.h>
#include <stdlib.h>
#include "inventory.h"

#ifndef Client
#define Client

Det bought;
int number, item_index;
char customer_name[MAX_SIZE];

Det get_item(int item_code)
{
   Det out;
   // Iterating through item list to get the item with given item code
   for (int i = 0; i < item_numbers; i++)
   {
```

```
        if (item[i].code = item_code)
        {
            out = item[i];
            item_index = i;
        }
    }
    return out;
}

void select_items()
{
    int selected_code = 0;
    fflush(stdout);
    printf("Enter your name please: ");
    scanf("%s", customer_name);
    printf("\nSelect by entering the item code.\n");
    display_stock();
    fflush(stdout);
    printf("\nEnter the code: ");
    scanf("%d", &selected_code);
    bought = get_item(selected_code); // Getting the selected item.
    printf("Enter how many %-15s items you want: ", bought.name);
    scanf("%d", &number); // getting the how many items to buy.
    double total_cost = bought.price * number; // computing the total price of the bought
items.
    printf("\nYou have successfully bought %d %s items for %.3lf amount\n", number,
bought.name, total_cost);
    item[item_index].qty -= number; // reducing the bough number of items from the stock.
}

#endif
```

seller.h

```
/*
The seller side library containing the functions that will allow the seller to access
all items in the stock, by adding and viewing available items in the stock.
*/
#include <stdio.h>
#include <stdlib.h>
#include "inventory.h"
#include "client.h"

#ifndef Seller
#define Seller

// Showing the seller all available items in the stock.
void available_items()
{
   printf("\nYour current stock status is:\n\n");
   display_stock();
}

// Showing the recent purchase from the customers.
void recent_purchase()
{
   printf("\nThe customer named %s bought the following  %d %s items\n\n",
customer_name, number, bought.name);
   fflush(stdout);
   available_items();
}

// Adding new items to the stock.
void add_items()
{
   int item_number;
   printf("\nHow many Items you want to add\n");
   scanf("%d", &item_number);
   int new_items = item_numbers + item_number;
   Det *new_list = (Det *)malloc(new_items * sizeof(Det)); // initializing a new array of
needed size
   // copying item elements to new created array.
```

```
    for (int j = 0; j < item_numbers; j++)
    {
        new_list[j] = item[j];
    }
    free(item);
    item = new_list; // extending item to new created array.
    new_list = NULL;
    int i;
    // Adding items to the stock
    for (i = item_numbers; i < new_items; i++)
    {
        fflush(stdout);
        printf("Item name: \n");
        scanf("%s", item[i].name);
        fflush(stdout);
        printf("Item code: \n");
        scanf("%d", &item[i].code);
        fflush(stdout);
        printf("Quantity: \n");
        scanf("%d", &item[i].qty);
        fflush(stdout);
        printf("price: \n");
        scanf("%f", &item[i].price);
        fflush(stdout);
        printf("Manufacturing date(dd-mm-yyyy): \n");
        scanf("%d-%d-%d", &item[i].mfg.day,
              &item[i].mfg.month, &item[i].mfg.year);
    }
    item_numbers = new_items; // updating number of items in the stock.
}

// Calculating the total cost of all items in the stock.
void total_cost()
{
    double total_cost = 0;
    for (int i = 0; i < item_numbers; i++)
        total_cost += item[i].price * item[i].qty;
    printf("You have %d items which costs: %.3lf\n", item_numbers, total_cost);
}
#endif
```

inventory.h

```
/*
The inventory library to keep track of our stock by initializing and adding items to the stock.
Two main functions here:
----------------------
initialize_stock(), The seller has to first initialize items in the stock for them to be available for clients.
display_stock(), It helps both client and seller to know current items in the stock.
*/
#include <stdio.h>
#include <stdlib.h>
#define MAX_SIZE 20

#ifndef Inventory
#define Inventory

// Storing the date information in a structure
struct date
{
   int day;
   int month;
   int year;
};

// Initializing a structure of item details and type defining it to Det.
typedef struct details
{
   char name[MAX_SIZE];
   float price;
   int code;
   int qty;
   struct date mfg;
} Det;

Det *item;        // Declaring a pointer that stores array of items.
int item_numbers; // global value to keep track of items we have in the array.
void initialize_stock()
{
   printf("\nEnter the number of items to initialize\n");
   scanf("%d", &item_numbers);
   item = (Det *)malloc(item_numbers * sizeof(Det)); // Initializing the size of array of items
   int i;
   // Getting items to register in the stock.
   for (i = 0; i < item_numbers; i++)
   {
```

```
        fflush(stdout);
        printf("Item name: \n");
        scanf("%s", item[i].name);
        fflush(stdout);
        printf("Item code: \n");
        scanf("%d", &item[i].code);
        fflush(stdout);
        printf("Quantity: \n");
        scanf("%d", &item[i].qty);
        fflush(stdout);
        printf("price: \n");
        scanf("%f", &item[i].price);
        fflush(stdout);
        printf("Manufacturing date(dd-mm-yyyy): \n");
        scanf("%d-%d-%d", &item[i].mfg.day,
              &item[i].mfg.month, &item[i].mfg.year);
    }
}

// Displaying the stock contents a table showing items in the stock with full details.+void
display_stock()
{
    int i;
    printf("                    *****  INVENTORY ***** \n");
    printf("--------------------------------------------------------------------\n");
    printf(" S.N.|     NAME     | CODE  | QUANTITY | PRICE  |   MFG.DATE \n");
    printf("--------------------------------------------------------------------\n");
    for (i = 0; i < item_numbers; i++)

printf("%d.     %-15s        %-d        %-5d      %.3f         %d/%d/%d \n", i + 1, item[i].name,
item[i].code, item[i].qty,
              item[i].price, item[i].mfg.day, item[i].mfg.month,
              item[i].mfg.year);
    printf("--------------------------------------------------------------------------------\n");
}

#endif
```

main.c

```
/*
The main program of our application
-----------------------------------
Including all the three header files,
and invoke the functions from header files according to the user choice.
We have two options to login as the client or seller.
But to get the better exerience it is mandatory for the seller to login first and initialize stock.
before performing any other operation and do other operatations accordingly without quiting.

*/
#include <stdio.h>
#include <stdlib.h>
#include "inventory.h"
#include "seller.h"
#include "client.h"

int main()
{
   printf("\nWelcome to the Vending Machine\n");
   int main_choice;
   do
   {

      printf("\nMenu:\n\n");
      printf("     1. Client     \n");
      printf("     2. Seller     \n");
      printf("     0. Quit     \n");
      printf("--------------------\n");
      scanf("%d", &main_choice);
      switch (main_choice)
      {
      case 1:
         printf("\nSuccessfully logged in as the Client.\n");
```

```
    select_items();
    break;
case 2:
    printf("\nSuccessfully logged in as the Seller.\n");
    int seller_choice;
    do
    {
        printf("\nSeller's Menu:\n\n");
        printf("     1. Initialize Stock    \n");
        printf("     2. Add items     \n");
        printf("     3. View total cost     \n");
        printf("     4. View available items     \n");
        printf("     5. To view recent purchase     \n");
        printf("-------------------------------------\n");
        scanf("%d", &seller_choice);
        switch (seller_choice)
        {
        case 1:
            initialize_stock();
            break;
        case 2:
            add_items();
            break;
        case 3:
            total_cost();
            break;
        case 4:
            available_items();
            break;
        case 5:
            recent_purchase();
            break;
        case 0:
            printf("\nLogging out\n");
            break;
```

```
            default:
                break;
            }
        } while (seller_choice <= 5 && seller_choice > 0);

        break;
    case 0:
        printf("\nThank you for using VENDICINE.\n");
        break;
    default:
        break;
    }
  } while (main_choice > 0 && main_choice < 3);

  return 0;
}
```

OUTPUT:

```
Menu:

     1. Client
     2. Seller
     0. Quit
--------------------
1

Successfully logged in as the Client.
Enter your name please: Felix

Select by entering the item code.
                      *****  INVENTORY *****
-----------------------------------------------------------------------
 S.N.|      NAME           |    CODE   |  QUANTITY |  PRICE    |     MFG.DATE
-------------------------------------------------------------------------------
1.     Coca                    800        23         20.000         9012022/0/0
2.     Sprite                  900        25         20.000         9082022/0/0
-------------------------------------------------------------------------------

Enter the code: 800
Enter how many Sprite          items you want: 5

You have successfully bought 5 Sprite items for 100.000 amount
```

PROJECT 2 – DATABASE MANAGEMENT FILE READER AND GENERATOR

DESCRIPTION:

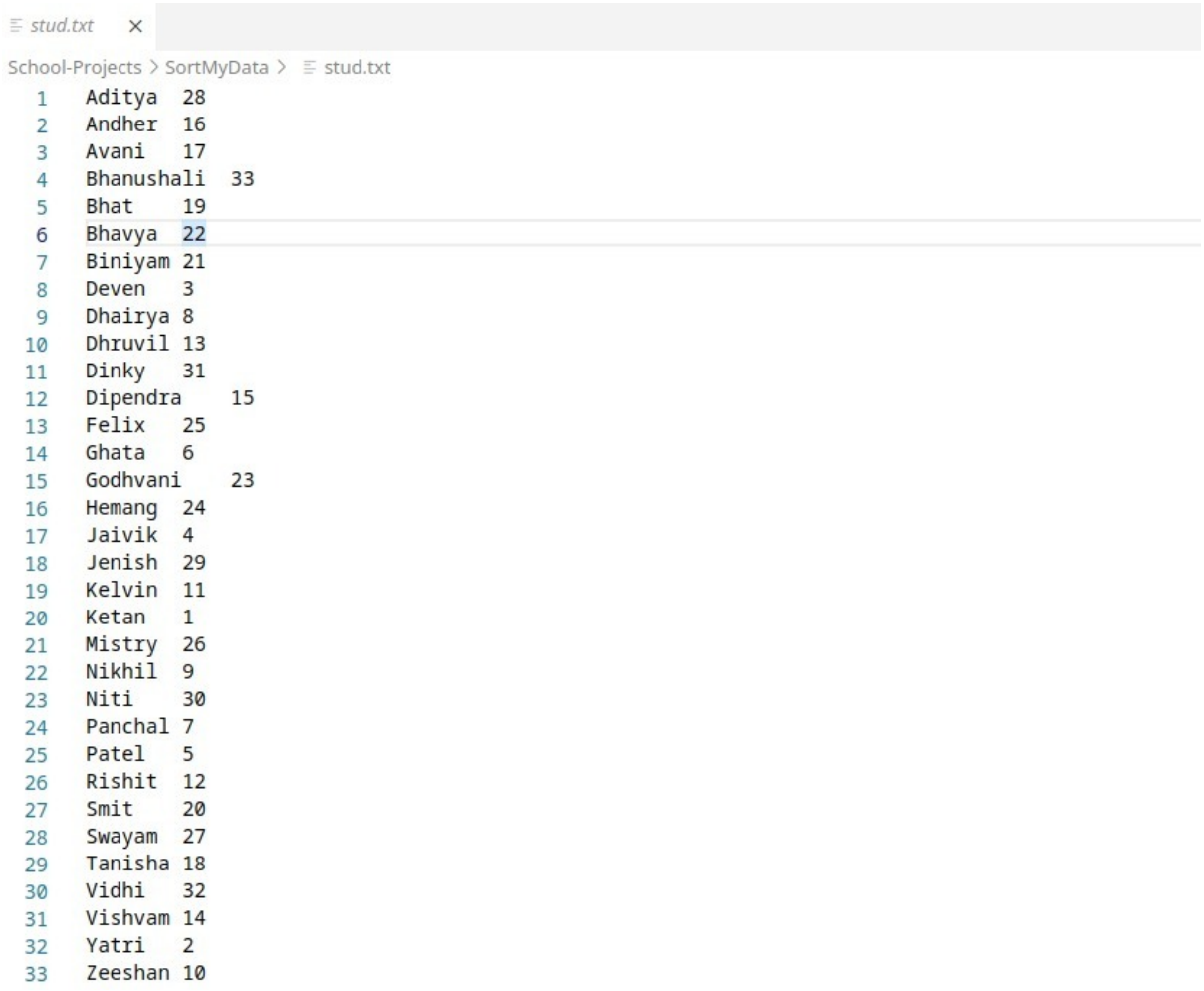
Used file operations to read data from .txt file and sort the data. After then write the sorted content to output file. This project uses file operations and Structures to read the student's data from the file sort them and give a new version of sorted data on the output file. There is one mail file named as SortText.c

CODE:

```
#include <stdio.h>
#include <stdlib.h>

typedef struct student
{
   char name[20];
   int rollno;
} ST;

ST stud[200];

void main()
{
   printf("Sorting the text file in C\n");
   printf("---------------------------");
   struct student temp;
   FILE *fp;
   int i = 0, size, j;
   char ch;
   printf("\nReading the file.....\n");
   fp = fopen("stud.txt", "r");
   if (fp == NULL)
   {
      printf("\n Cannot open the file \n");
```

```
        exit(0);
    }
    while (ch != EOF)
    {
        fscanf(fp, "%s%d", stud[i].name, &stud[i].rollno);
        ch = fgetc(fp);
        i++;
    }
    printf("Sorting based on Roll Numbers.....\n");
    size = i - 1;
    for (i = 1; i < size; ++i)
        for (j = 0; j < size - i; j++)
            if (stud[j + 1].rollno < stud[j].rollno)
            {
                temp = stud[j];
                stud[j] = stud[j + 1];
                stud[j + 1] = temp;
            }
    printf("Writting sorted Data to a text file.....\n");
    fp = fopen("SortedData.txt", "w");
    for (i = 0; i < size; i++)
        fprintf(fp, "%s %d \n", stud[i].name, stud[i].rollno);
    printf("\nThe file is sorted successfully and saved as SortedData.txt. \n \n");
}
```

OUTPUT:


stud.txt
School-Projects > SortMyData > stud.txt
1 Aditya 28
2 Andher 16
3 Avani 17
4 Bhanushali 33
5 Bhat 19
6 Bhavya 22
7 Biniyam 21
8 Deven 3
9 Dhairya 8
10 Dhruvil 13
11 Dinky 31
12 Dipendra 15
13 Felix 25
14 Ghata 6
15 Godhvani 23
16 Hemang 24
17 Jaivik 4
18 Jenish 29
19 Kelvin 11
20 Ketan 1
21 Mistry 26
22 Nikhil 9
23 Niti 30
24 Panchal 7
25 Patel 5
26 Rishit 12
27 Smit 20
28 Swayam 27
29 Tanisha 18
30 Vidhi 32
31 Vishvam 14
32 Yatri 2
33 Zeeshan 10

PROJECT 3- RESTAURANT MANAGEMENT

DESCRIPTION:

We used the array to implement a queue with different functionalities that are used in a queue. These are the operations that are implemented in the queue: create, insert, delete, display, and exit.

We used the queue to implement a restaurant management system where we can add customers to the queue, delete, display the queue and exit the program. Also, this can find applications in car parking systems, hospitals, and many other places where we need to manage a queue of people.

CODE:

```
#include <stdio.h>
#include <conio.h>
#define SIZE 10

void insert(int);
void delete();
void display();

int queue[SIZE], front = -1, rear = -1;

void main()
{
   int value, choice;
   // clrscr();
   while (1)
   {
      printf("\n1 - Enter your ID number: ");
      printf("\n2 - Enter ID number to be deleted: ");
      printf("\n3 - Display all ID numbers: ");
      printf("\n4 - Exit");

      // create();
      printf("\nEnter your choice : ");
      scanf("%d", &choice);
      switch (choice)
      {
      case 1:
         printf("\nEnter your ID number: ");
         scanf("%d", &value);
         insert(value);
         break;
      case 2:
         printf("\nEnter ID number to be deleted: ");
         scanf("%d", &value);
```

```
            delete();
            break;
        case 3:
            display();
            break;
        // case 4:
        //     exit(0);
        default:
            printf("\nChoice is incorrect, Enter a correct choice");
        }
    }
}
void insert(int value)
{
    if (rear == SIZE - 1)
        printf("\nQueue overflow no more elements can be inserted");
    else
    {
        if (front == -1)
            front = 0;
        rear++;
        queue[rear] = value;
        printf("Customer is added successfully\n");

    }
}
void delete()
{
    if (front == rear)
        printf("\\nQueue is empty no elements to delete");
    else
    {
        printf("\nCustomer key deleted is : %d", queue[front]);
        front++;
        if (front == rear)
            front = rear = -1;
    }
}
void display()
{
    if (rear == -1)
        printf("\nQueue is Empty!!!");
    else
    {
        int i;
        for (i = front; i <= rear; i++)
            printf("Elements are :%d\n", queue[i]);
    }
}
```

OUTPUT:

```
1 - Enter your ID number:
2 - Enter ID number to be deleted:
3 - Display all ID numbers:
4 - Exit
Enter your choice : 1

Enter your ID number: 4
Customer is added successfully
Enter your ID number: 5
Customer is added successfully

1 - Enter your ID number:
2 - Enter ID number to be deleted:
3 - Display all ID numbers:
4 - Exit
Enter your choice : 3
Elements are :4
Elements are :5

1 - Enter your ID number:
2 - Enter ID number to be deleted:
3 - Display all ID numbers:
4 - Exit
Enter your choice : 2

Enter ID number to be deleted: 5

Customer key deleted is : 4
```

PROJECT 4- MUSIC PLAYER

DESCRIPTION:
Simulation of how music player or playlist works using a linked list where we can add songs to the playlist play them and delete them. The project was designed using a Doubly Circular Linked Lists to mimic the playlist behaviour where each node represents a certain song, and all the list represents the playlist. When playing the songs, the time duration of a song is respected before jumping to the next song. There are 3 header files named as insertion.h, playing.h, song.h and one main file namely main.c

CODE:

insertion.h

```
/*
The header file for insertion and deletion of songs from the playlist
*/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "song.h"

#ifndef DEL
#define DEL

// Inserting a song at the end of the playist
void insert()
{
  printf("Enter name of the song: \n");
  while (getchar() != '\n')
    ;
  scanf("%[^\n]%*c", temp);
  printf("\nEnter the song duration: ");
  scanf("%u", &song_time);
  Song *new_node = (Song *)malloc(sizeof(Song)); // Allocate memory for new node
  strcpy(new_node->song, temp);
  new_node->song_time = song_time;
  // Adding the new node to current node
  if (head == NULL)
  {
    new_node->next = new_node->previous = new_node;
    head = current_node = new_node;
    return;}
  struct node *last = head->previous;
```

```
  new_node->previous = last;
  last->next = new_node;
  new_node->next = head;
  head->previous = new_node;
};

// Deleting a song at the end of the playist
void delete_song()
{
  if (head == NULL)
  {
    printf("There is no in this playlist!!\n");
    return;
  }
  printf("\nAvailable songs in the playlist\n");
  display();
  printf("\nEnter name of the song need to be deleted: ");
  while (getchar() != '\n')
    ;
  scanf("%[^\n]%*c", temp);
  Song *ptr = head;
  // check the song from the playlist
  do
  {
    if (ptr->next == ptr && strcmp(ptr->song, temp) == 0)
    {
      printf("ONE SONG DELETED\n");
      printf("PLAYLIST IS EMPTY NOW.");
      head == NULL;
      free(ptr);
      return;
    }
    else if (strcmp(ptr->song, temp) == 0)
    {
      struct node *previous = ptr->previous;
      struct node *next = ptr->next;
      previous->next = next;
      next->previous = previous;
      head = next;
      free(ptr);
      printf("SONG IS DELETED SUCCESSFULLY\n");
      return;
    }
    ptr = ptr->next;
  } while (ptr != head);
  printf("The indicated song is not present\n");
} #endif
```

song.h

```
/*
The header file for storing the Song information.
*/
#include <stdio.h>
#include <stdlib.h>
#define MAX_SIZE 100

#ifndef SONG
#define SONG

// defining the song structure which will be called as node
// to the linked list

typedef struct node
{
   char song[MAX_SIZE];
   struct node *next;
   struct node *previous;
   unsigned int song_time;
} Song;

// Declaring variable that will store our songs information

char temp[MAX_SIZE];
unsigned int song_time;
Song *head;
Song *current_node;
int song_count;

// Initializing the song pointers
void initialize()
{
   head = NULL;
   current_node = NULL;
};

// Displaying the playlist contents.
void display()
{
   if (head == NULL)
   {
   printf("THE PLAYLIST IS EMPTY!!\n");
   return;
   }
   Song *display_ptr = head;
   int i = 1;
   printf("DISPLAYING PLAYLIST: \n");
```

```
    printf("\nSr.No| SONG NAME | Duration\n");
    printf("-----------------------------\n");
    do
    {
    printf(" %d       %s            %u\n", i, display_ptr->song, display_ptr->song_time);
    i++;
    display_ptr = display_ptr->next;
    } while (display_ptr != head);
};

// Counting the number of songs available in the playlist.
void count_songs()
{
    if (head == NULL)
    {
      printf("THE PLAYLIST IS EMPTY!!\n");
      return;
    }
    Song *songs = head;
    int i = 1;
    do
    {
      i++;
      songs = songs->next;
    } while (songs != head);
    song_count = i-1;
};
#endif
```

playing.h

```
/*
The header file for playing functionalities of the Music Player.
*/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "song.h"
#include <time.h>

#ifndef PLAY
#define PLAY

// Delay for playing the song
void time_delay(unsigned int song_time)
{
  unsigned int retTime = time(0) + song_time;
  while (time(0) < retTime)
    ;
}

// Playing the song from the playlist
void play()
{
  if (head == NULL)
  {
    printf("THE PLAYLIST IS EMPTY!!\n");
    return;
  }
  Song *play_song = head;
  int i = 1;
  printf("\nPlaying the songs from the playlist\n\n");
  printf("Current Playing|  Duration\n");
  printf("--------------------------\n");
  do
  {
    printf(" %s              %u\n", play_song->song, play_song->song_time);
    time_delay(play_song->song_time);
    i++;
    play_song = play_song->next;
  } while (play_song != head);
}

#endif
```

main.c

```
#include <stdio.h>
#include <stdlib.h>
#include "song.h"
#include "insertion.h"
#include "playing.h"

int main()
{
    printf("\nWelcome to the Music player!!!\n");
    initialize();
    int select;
selection:
    printf("\n\n--------MENU--------\n");
    printf("\n1. ADD A SONG\n");
    printf("\n2. REMOVE A SONG\n");
    printf("\n3. DISPLAY PLAYLIST\n");
    printf("\n4. PLAY ALL SONGS\n");
    printf("\n5. COUNT ALL SONGS\n");
    printf("\n6. EXIT\n");
    printf("---------\n");

    scanf("%d", &select);

    if (select == 1)
        insert();

    else if (select == 2)
        delete_song();
```

```
    else if (select == 3)
        display();

    else if (select == 4)
        play();
    else if (select == 5)
    {
        count_songs();
        printf("\n TOTAL SONGS - %d ", song_count);
    }
    else
        goto thankyou;
    goto selection;
thankyou:
    system("clear");
    printf("\nThank you for using our application");
    printf("\u263A\n");
    exit(0);
```

OUTPUT:

```
__

Welcome to the Music player!!!


--------MENU--------

1. ADD A SONG

2. REMOVE A SONG

3. DISPLAY PLAYLIST

4. PLAY ALL SONGS

5. COUNT ALL SONGS

6. EXIT
---------
1
Enter name of the song:
Red flags

Enter the song duration: 23
```

```
--------MENU--------

1. ADD A SONG

2. REMOVE A SONG

3. DISPLAY PLAYLIST

4. PLAY ALL SONGS

5. COUNT ALL SONGS

6. EXIT
---------
4

Playing the songs from the playlist

Current Playing|  Duration
---------------------------
 Red flags               23
 Face off               15
```

PROJECT 5- STORE MANAGEMENT

DESCRIPTION:

Using Binary Search Tree to manage store items. In this Project we used the Binary Search tree to organise store Items which makes it easier for the store manager to find certain items given he knows the price as all items are organised price wise given that the first item to be in store is considered the Tree root node. It helps to know total cost, display all items and search through the store. There are three header files named as items.h, manager.h, database.h and two main files main.c and nodec.c

CODE:

items.h

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define MAX 20

typedef struct BST
{
   int price;
   char *name;
   struct BST *left;
   struct BST *right;
} node;

node *newNode(int price, char *name)
{
   node *temp = (node *)malloc(sizeof(node));
   temp->price = price;
   temp->name = name;
   temp->left = NULL;
   temp->right = NULL;
   return temp;
}

// A utility function to do inorder traversal of BST
void inorder(node *root)
{
   if (root != NULL)
   {
```

```
        inorder(root->left);
        printf("Price: %d, Item: %s\n", root->price, root->name);
        inorder(root->right);
    }
}

// A utility function to print preorder traversal of BST
void postorder(node *root)
{
    if (root != NULL)
    {
        postorder(root->left);
        postorder(root->right);
        printf("Price: %d, Item name: %s\n", root->price, root->name);
    }
}

// A utility function to print preorder traversal of BST
void preorder(node *root)
{
    if (root != NULL)
    {
        preorder(root->right);
        preorder(root->left);
        printf("Price: %d, Item name: %s\n", root->price, root->name);
    }
}

// A utility function to find minimum value node in BST
node *minValueNode(node *to_search)
{
    node *current = to_search;
    while (current && current->left != NULL)
        current = current->left;
    return current;
}

// A utility function to find maximum value node in BST
node *maxValueNode(node *to_search)
{
    node *current = to_search;
    while (current && current->right != NULL)
        current = current->right;
    return current;
}
```

manager.h

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "items.h"
#include "database.h"

/*Predefined functions*/
char *file = "file.txt";
void update_database(node*curr);
void display_all_elemtents(node*root, int disp_mode);

// A utility function to determine whether a node is present in BST
int search(node*root, int search_price)
{
   if (root == NULL)
      return 0;
   if (root->price == search_price)
      return 1;
   if (root->price > search_price)
      return search(root->left, search_price);
   return search(root->right, search_price);
}

// A utility function to insert a new node with given price in BST
void insert_node_to_tree(node*root, int to_ins_price, char *ins_name)

{
   node*to_insert = newNode(to_ins_price, ins_name);
   if (to_insert->price < root->price)
   {
      if (root->left != NULL)
         insert_node_to_tree(root->left, to_insert->price, to_insert->name);
      else
         root->left = to_insert;
   }
   if (to_insert->price > root->price)
   {
      if (root->right != NULL)
         insert_node_to_tree(root->right, to_insert->price, to_insert->name);
      else
         root->right = to_insert;
   }
   delete_file_content(file);
   update_database(root);
   // return root;
}
```

```
// A utility function to add items to the database
node*add_items(node*root, char *name, int price)
{
   node*new_node = (node*)malloc(sizeof(node));
   new_node->name = name;
   new_node->price = price;
   new_node->left = NULL;
   new_node->right = NULL;

   if (root == NULL)
   {
      root = new_node;
   }
   else
   {
      node*current_node = root;
      while (current_node != NULL)
      {
         if (price < current_node->price)
         {
            if (current_node->left != NULL)
            {
               current_node = current_node->left;
            }
            else
            {
               current_node->left = new_node;
               break;
            }
         }
         else if (price > current_node->price)
         {
            if (current_node->right != NULL)
            {
               current_node = current_node->right;
            }
            else
            {
               current_node->right = new_node;
               break;
            }
         }
      }
   }

   delete_file_content(file);
   update_database(root);
```

```
    return root;
}
// A utility function to delete from a BST
node*delete_node_from_tree(node*root, int del_price)
{

    if (search(root, del_price) == 0)
    {
        printf("\nItem not found in the tree\n");
        return root;
    }
    if (root == NULL)
        return root;
    if (del_price < root->price)
        root->left = delete_node_from_tree(root->left, del_price);
    else if (del_price > root->price)
        root->right = delete_node_from_tree(root->right, del_price);
    else
    {
        if (root->left == NULL)
        {
            node*temp = root->right;
            free(root);
            return temp;
        }
        else if (root->right == NULL)
        {
            node*temp = root->left;
            free(root);
            return temp;
        }
        node*temp = minValueNode(root->right);
        root->price = temp->price;
        root->right = delete_node_from_tree(root->right, temp->price);
    }
    delete_file_content(file);
    update_database(root);
    return root;
}

void display_all_elemtents(node*root, int disp_mode)
{
    if (disp_mode == 1)
        inorder(root);
    else if (disp_mode == 2)
        preorder(root);
    else if (disp_mode == 3)
        postorder(root);
```

```
    else
        printf("Invalid display mode");
}

// A utility function to find sum of all elements in BST
int sum_of_all_elements(node*root)
{
    if (root == NULL)
        return 0;
    return (root->price + sum_of_all_elements(root->left) + sum_of_all_elements(root-
>right));
}

char *get_item_name(node*root, int search_price)
{
    if (root == NULL)
        return NULL;
    if (root->price == search_price)
        return root->name;
    if (root->price > search_price)
        return get_item_name(root->left, search_price);
    return get_item_name(root->right, search_price);
}

// Update database function
void update_database(node*curr)
{
    if (curr != NULL)
    {
        update_database(curr->left);
        char *price_conv = convert_int_to_string(curr->price);
        char *nam = get_item_name(curr, curr->price);
        char *to_write = concatenate_strings(price_conv, nam);
        write_single_line(file, to_write);
        update_database(curr->right);
    }
}
```

database.h

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>

// Utility functions to access the database

// Function to read the database from the file
char **read_line_by_line(char *filename)
{
   FILE *fp;
   char **lines = NULL;
   char *line = NULL;
   size_t len = 0;
   ssize_t read;
   int i = 0;

   fp = fopen(filename, "r");
   if (fp == NULL)
      exit(EXIT_FAILURE);

   while ((read = getline(&line, &len, fp)) != -1)
   {
      lines = realloc(lines, sizeof(char *) * (i + 1));
      lines[i] = malloc(len);
      strcpy(lines[i], line);
      i++;
   }

   fclose(fp);
   if (line)
      free(line);
   return lines;
}

// Function to split a string into tokens
char **split_string(char *line)
{
   char **tokens = NULL;
   char *token = NULL;
   int i = 0;

   token = strtok(line, " ");
   while (token != NULL)
   {
```

```
        tokens = realloc(tokens, sizeof(char *) * ++i);
        tokens[i - 1] = token;
        token = strtok(NULL, " ");
    }
    tokens = realloc(tokens, sizeof(char *) * (i + 1));
    tokens[i] = 0;
    return tokens;
}

// Function to convert integer string to integer
int convert_string_to_int(char *string)
{
    int i = 0;
    int number = 0;
    while (string[i] != '\0')
    {
        number = number * 10 + (string[i] - '0');
        i++;
    }
    return number;
}

// Function to convert integer to string
char *convert_int_to_string(int number)
{
    char *string = malloc(10);
    sprintf(string, "%d", number);
    return string;
}

// Function to concatenate two strings with a space in between
char *concatenate_strings(char *string1, char *string2)
{
    char *string = malloc(strlen(string1) + strlen(string2) + 2);
    strcpy(string, string1);
    strcat(string, " ");
    strcat(string, string2);
    return string;
}

// Function to write the database to the file
void write_single_line(char *filename, char *line)
{
    FILE *fp;
    fp = fopen(filename, "a");
    if (fp == NULL)
        exit(EXIT_FAILURE);
    fprintf(fp, "%s \n", line);
```

```
    fclose(fp);
}

// Delete all file content
void delete_file_content(char *filename)
{
    FILE *fp;
    fp = fopen(filename, "w");
    if (fp == NULL)
        exit(EXIT_FAILURE);
    fclose(fp);
}
```

store.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "manager.h"

int main()
{
   printf("\nWelcome to the Store Management App!!!\n");
   int select;
selection:
   printf("\n\n--------MENU--------\n");
   printf("\n1. Add Items\n");
   printf("\n2. Display All Items\n");
   printf("\n3. Expensive Or Cheap Item\n");
   printf("\n4. Delete Item\n");
   printf("\n5. Calculate Total Cost of Store\n");
   printf("\n6. Exit\n");
   printf("---------\n");
   printf("Enter your choice: ");
   scanf("%d", &select);
   node *root;
   if (select == 1)
   {
      printf("\nWhat is the name of your Item to add?: ");
      char name[MAX];
      scanf("%s", name);
      printf("\nHow Much is your Item to add?: ");
      int price;
      scanf("%d", &price);
      node *new = add_items(root, name, price);
      root = new;
      display_all_elemtents(new, 1);
   }

   else if (select == 2)
   {
      printf("\nSelect\n%d to display inorder,\n%d to display preorder,\n%d to display
postorder\n: ", 1, 2, 3);
      int choice;
      scanf("%d", &choice);
      display_all_elemtents(root, choice);
   }
   else if (select == 3)
   {
      printf("\nselect %d to display expensive item,\n%d to display q cheap item: ", 1, 2);
      int choice;
```

```
        scanf("%d", &choice);
        node *item;
        if (choice == 1)
        {
            item = maxValueNode(root);
            printf("\nexpensive item is %s with price %d\n", item->name, item->price);
        }
        else if (choice == 2)
        {
            item = minValueNode(root);
            printf("\ncheap item is %s with price %d\n", item->name, item->price);
        }
    }
    else if (select == 4)
    {
        printf("\nHow Much is your Item to delete?: ");
        int price;
        scanf("%d", &price);
        node *temp = delete_node_from_tree(root, price);
        printf("\nItem deleted successfully!!!\n");
        root = temp;
    }
    else if (select == 5)
    {
        int total = sum_of_all_elements(root);
        printf("\nThe total cost of your store is: %d\n", total);
    }
    else
        goto thankyou;
    goto selection;
thankyou:
    printf("\nThank you for using our application");
    printf("\u263A\n");
    exit(0);
}
```

nodec.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "manager.h"

int main()
{
    node*root = NULL;
    printf("\nFirst\n");
    add_items(root, "nshs", 50);
    printf("\nshmd\n");
    add_items(root, "ks", 100);
    // printf("Hello");
    display_all_elemtents(root, 1);
    return 0;
}
```

OUTPUT:

```
Welcome to the Store Management App!!!


--------MENU--------

1. Add Items

2. Display All Items

3. Expensive Or Cheap Item

4. Delete Item

5. Calculate Total Cost of Store

6. Exit
---------
Enter your choice: 1

What is the name of your Item to add?: Coke

How Much is your Item to add?: 100
Price: 100, Item: Coke
Price: 100, Item: Coke
```

```
6. Exit
---------
Enter your choice: 5

The total cost of your store is: 120


--------MENU--------

1. Add Items

2. Display All Items

3. Expensive Or Cheap Item

4. Delete Item

5. Calculate Total Cost of Store

6. Exit
---------
Enter your choice: 2

Select
1 to display inorder,
2 to display preorder,
3 to display postorder
: 3
Price: 20, Item name: Sprite
Price: 100, Item name: Sprite
```

PROJECT 6- GRAPHICAL REPRSENTATION OF SHORTEST PATH PROBLEM

DESCRIPTION:

Dijkstra algorithm is a single-source shortest path algorithm. Here, single source means that only one source is given, and we have to find the shortest path from the source to all the nodes. Integration of the **matplotlib** from python libraries has been done to plot the shortest path of the matrices. There is one file containing main code, one file contains the integration of python library.

CODE:

main.c

```
#include <limits.h>
#include <stdbool.h>
#include <stdio.h>
#define V 9

int minDistance(int dist[], bool sptSet[])
{
   int min = INT_MAX, min_index;
   for (int v = 0; v < V; v++)
      if (sptSet[v] == false && dist[v] <= min)
         min = dist[v], min_index = v;
   return min_index;
}

void printSolution(int dist[])
{
   printf("Vertex \t\t Distance from Source\n");
   for (int i = 0; i < V; i++)
      printf("%d \t\t\t\t %d\n", i, dist[i]);
}

void dijkstra(int graph[V][V], int src)
{
   int dist[V];
   bool sptSet[V];
   for (int i = 0; i < V; i++)
      dist[i] = INT_MAX, sptSet[i] = false;
   dist[src] = 0;
   for (int count = 0; count < V - 1; count++) {
```

```
        int u = minDistance(dist, sptSet);
        sptSet[u] = true;
        for (int v = 0; v < V; v++)
            if (!sptSet[v] && graph[u][v]
                && dist[u] != INT_MAX
                && dist[u] + graph[u][v] < dist[v])
                dist[v] = dist[u] + graph[u][v];
    }
    printSolution(dist);
}

int main()
{
    int graph[V][V] = {{1, 2},
         {5, 6},
         {9, 10,},
         {13, 14, 15, 16}};
    dijkstra(graph, 0);
    return 0;
}
```

integration.c

```
#include <Python.h>

int main(int argc, char *argv[])
{
   wchar_t *program = Py_DecodeLocale(argv[0], NULL);
   if (program == NULL) {
      fprintf(stderr, "Fatal error: cannot decode argv[0]\n");
      exit(1);
   }
   Py_SetProgramName(Graph);  /* optional but recommended */
   Py_Initialize();
   PyRun_SimpleString("from time import time,ctime,sleep\n"
                      "import matplotlib.pyplot as plt\n"
                      "import matplotlib\n"
                      "print(matplotlib.get_backend())\n"
                      "plt.ion()\n"
                      "plt.plot([1,2,3,4,3,5,7])\n"
                      "plt.pause(5)\n");
   Py_Finalize();
   PyMem_RawFree(Graph);
   return 0;
}
```

- To plot graph the code is as follows:

```
import "matplotlib.pyplot as plt"
import numpy as np

xpoints = np.array([1,5,9,13])
ypoints = np.array([2,6,10,16])

plt.plot(xpoints, ypoints)
plt.show()
```

OUTPUT: