Name: Mean Diamand
Neptun Code: KSG25Z

Developer Manual Of

Basic of Programming 1

Homework Project

Birthday and Anniversary

My program is operating on the window command prompt. This program is consisting of main.c file which contains the source code of my project which has all the necessary libraries, structures, sub-functions, and main functions.

In my database program, I included four libraries of c programming language:

- stdio.h for fopen(), fclose(), fscanf(), and fprintf()
- stdlib.h for system(), malloc(), and exit()
- string.h for strcpy()
- time.h for time_t, and time()

My Birthday and Anniversary Database program is consisting of:

- 3 Structures:
    1. **name** - used for defining a data type for storing first name and surname of the users by declaring character string data type structure elements called f_name(for the first name) and s_name(for the surname).

       ```
       typedef struct name
       {
          char f_name[100];
          char s_name[100];
       }name;
       ```

    2. **date** - used for defining new data types for storing the year, month, day of the date by declaring integer data type structure elements called year, month, and day.

       ```
       typedef struct date
       {
          int year;
          int month;
          int day;
       }date;
       ```

    3. **list** - used for defining a list of the name of the users or the husband and wife if they are a couple, category, type, and date of the users by declaring a structure element called 'n' and 'w'(for the first name and surname) as the name data type from the name structure, character string data type structure elements called category(for the category of the users. Ex: Friend, Relative, Family, Colleague…etc.), type (for the type of day. Ex: Birthday, Nameday, Anniversary, Marriage…etc.), and a structure element called 'd' (for the year, month, and day) as the date data type from the date structure. This structure also stores the address of the list as well.

       ```
       typedef struct list
       {
       ```

```
    name n;
    name w;
    char category[100];
    char type[100];
    date d;
    struct list *nxt;
}list;
```

- 19 Functions:

1. **read_the_file()** is a void function that is used for reading the text file that is wanted to be displayed on the console by using the file handling method for reading the text file.

```
void read_the_file(char *file)
{
    char c;
    FILE *fp=fopen(file, "r");
    if(fp==NULL)
    {
        return;
    }

    //reading character one by one until it is successful
    while(fscanf(fp, "%c", &c)==1)
    {
        printf("%c", c);
    }

    fclose(fp); //close file
    printf("\nInput 6 to go back\n\n");
    int n=1;
    while(n!=6)
    {
        scanf("%d", &n);
        if(n!=6)
        {
            printf("Please input the valid option!!!\n");
        }
    }
}
```

2. **introduction()** is an int function that returns the input choice of the users when they wanted to select the option in the menu, and it is also used for reading a text file that contains the introduction of my program. In this function, I implemented a file-handling method for reading a text file.

```
int introduction()
{

    system("cls");
    char c;
    FILE *fp=fopen("Introduction.txt", "r"); // open the text file
    if(fp==NULL)
    {
        return 1;
```

```
    }

    // reading character one by one until it is successful
    while(fscanf(fp, "%c", &c)==1)
    {
        printf("%c", c);
    }

    fclose(fp); // close file
    int choice;
    printf("\n\nPlease input your option: ");
    scanf("%d", &choice);
    return choice;
}
```

3. **databases()** is a void function that is used to choose the options which are displayed on the databases menu that will be shown when the users start the databases. In this function, I implemented the switch case in the conditional statement method for the database options.

```
void databases()
{
    int numOfPeople = 0;
    list* head = read_database(&numOfPeople);
    int opd = 0;
    while(opd!=5)
    {
        system("cls");
        printf("_____\n");
        printf("___ __ \\__   |__ __/__   |__ __)__   |_ ___/_ ____/_ ___/\n");
        printf("__ / / /_ /| |_ /  __ /| |_  __ |_ /| |___ \\__ __/ _____ \\ \n");
        printf("_ /_/ /_ ___| /  _ ___| /_/ /_ ___|___/ /_ /___ ____/ / \n");
        printf("/_____/ /_/ |_/_/   /_/ |_/_____/ /_/ |_/___/ /_____/ /___/  \n");
        printf("\n  1. Entering the data into the Databases\n");
        printf("  2. Searching for the data in the Databases\n");
        printf("  3. Deleting the data in the Databases\n");
        printf("  4. Printing the Databases\n");
        printf("  5. Back\n");
        scanf("%d", &opd);
        switch(opd)
        {
            case 1:
                head=data_entry(head, &numOfPeople);
                break;
            case 2:
                data_search(head);
                break;
            case 3:
                head=data_delete(head, &numOfPeople);
                break;
            case 4:
                print_databases(head);
                break;
            default:
                printf("Please choose the available options\n");
        }
    }
    save_database(head, numOfPeople);
}
```

Name: Mean Diamand
Neptun Code: KSG25Z

4. **data_entry()** is a function that receives a list, an address of an integer, and returns the list, I used this function for entering the input information from the users into the database. In this function, I declare a list-type pointer called p and allocate its memory space of it. Then, let the users input all the information into the node.

```
list *data_entry(list* head, int* numOfPeople)
{
   system("cls");
   list* p = (list*)malloc(sizeof(list));
   printf("Input the data into the Databases:\n");
   int td=5;
   while(td>4 || td<1)
   {
      printf("Type of day:\n");
      printf("1. Birthday\n");
      printf("2. Anniversary\n");
      printf("3. Nameday\n");
      printf("4. Marriage\n");
      scanf("%d", &td);
      switch(td)
      {
         case 1:
            strcpy(p -> type, "Birthday");
            break;
         case 2:
            strcpy(p -> type, "Anniversary");
            break;
         case 3:
            strcpy(p -> type, "Nameday");
            break;
         case 4:
            strcpy(p -> type, "Marriage");
            break;
         default:
            printf("Please choose the available options\n");
            break;
      }
   }

   int ctr=5;
   while(ctr>4 || ctr<1)
   {
      printf("Category of the users:\n");
      printf("1. Friend\n");
      printf("2. Family\n");
      printf("3. Relative\n");
      printf("4. Colleague\n");
      scanf("%d", &ctr);
      switch(ctr)
      {
         case 1:
            strcpy(p -> category, "Friend");
            break;
         case 2:
            strcpy(p -> category, "Family");
            break;
         case 3:
            strcpy(p -> category, "Relative");
            break;
         case 4:
            strcpy(p -> category, "Colleague");
            break;
         default:
            printf("Please choose the available options\n");
            break;
      }
```

```
    }

    printf("Name(Firstname Surname): ");
    scanf("%s %s", p -> n.f_name, p -> n.s_name);
    if(td==2 || td==4)
    {
        printf("Partner's Name(Firstname Surname): ");
        scanf("%s %s", p->w.f_name, p->w.s_name);
    }
    else
    {
        strcpy(p->w.f_name, "None");
        strcpy(p->w.s_name, "None");
    }

    printf("Input the Date (Ex: 2000 01 01):\n");
    printf("Date(YYYY MM DD): ");
    scanf("%d %d %d", &(p -> d.year), &(p -> d.month), &(p -> d.day));
    p -> nxt = NULL;

    head = insert_data(head, p);
    (*numOfPeople) ++;

    // hold the output
    int b = 0;
    while (b != 4) {
        printf("\n\nEnter 4 to go back\n");
        scanf("%d", &b);
    }
    free(p);
    return head;
}
```

5. **read_database()** is a function that receives the address of the integer and returns the list, I used this function for reading the data that were already inside the text file into the list. For example, if we want to use a text file that contained all the information, but that information wasn't contained in the list then this function will read all that information into the list.

```
list* read_database(int* numOfPeople)
{
    list* head = NULL;
    FILE* fp = fopen("History.txt", "r");
    fscanf(fp, "%d\n", numOfPeople);
    for (int i=0; i<*numOfPeople; i+=1)
    {
        list*   p = (list*) malloc (sizeof(list));
        fscanf(fp, "%s %s | %s %s | %s | %s | %d/%d/%d\n", p->n.f_name, p->n.s_name, p->w.f_name, p->w.s_name, p->category, p->type, &(p->d.year), &(p->d.month), &(p->d.day));
        head = insert_data(head, p);
        free(p);
    }
    fclose(fp);
    return head;
}
```

6. **save_database()** is a void function that receives a list and an integer value, I used this function for saving the new list that has been modified by the user into the text file. It is the function used for overwriting the data saved into the text file after the program has terminated.

```
void save_database(list* head, int numOfPeople)
{
    FILE* fp = fopen("History.txt", "w");
    if (fp == NULL)
    {
        printf("cannot open file to save\n");
        return;
```

```
    }
    fprintf(fp, "%d\n", numOfPeople);

    if(head == NULL)
    {
        return;
    }
    list* p = head;
    list* q = head -> nxt;

    for (int i = 0; i < numOfPeople; i+=1)
    {
        fprintf(fp, "%s %s | %s %s | %s | %s | %d/%d/%d\n", p->n.f_name, p->n.s_name, p->w.f_name, p->w.s_name, p->category, p->type, p->d.year, p->d.month, p->d.day);

        free(p);
        p = q;

        if (q != NULL) {
            q = q -> nxt;
        }

    }
    fclose(fp);
}
```

7. **insert_data()** is a function that receives two lists and then returns the new list, I used this function for inserting the node into the list. First, I declare a list pointer type called u. Then, I give a condition if the head is NULL which means if the list is empty then the program will call the push_front() function to add the first node to this empty list. After that, I create a loop to move u to the last node of the list. Then, I allocate the memory space for the node after the last node. After that, I use the strcpy() function to copy the string from personToAdd to the new node in the list and let the year, month, and day of the new node be equal to the year, month, and day of personToAdd. Finally, I let the last node point to NULL since it doesn't have any node to point to.

```
list *insert_data(list *head, list* personToAdd)
{
    list *u;
    if(head == NULL)
    {
        return push_front(head, personToAdd);
    }
    for(u=head; u->nxt!=NULL; u=u->nxt);

    u->nxt = (list*)malloc(sizeof(list));

    strcpy(u -> nxt -> n.f_name, personToAdd -> n.f_name);
    strcpy(u -> nxt -> n.s_name, personToAdd -> n.s_name);
    strcpy(u -> nxt -> w.f_name, personToAdd -> w.f_name);
    strcpy(u -> nxt -> w.s_name, personToAdd -> w.s_name);
    strcpy(u -> nxt -> category, personToAdd -> category);
    strcpy(u -> nxt -> type, personToAdd -> type);
    u -> nxt -> d.year = personToAdd -> d.year;
    u -> nxt -> d.month = personToAdd -> d.month;
    u -> nxt -> d.day = personToAdd -> d.day;
    u -> nxt -> nxt = NULL;

    return head;
}
```

8. **push_front()** is a function that receives two lists and returns the new list, I used this function for adding the first node into an empty list. First, I declare a list-type pointer called p and allocate its memory space of it. After that, I use the strcpy() function to copy the string from personToAdd to the new node in the list and let the year, month, and day of the new node be

equal to the year, month, and day of personToAdd. Then, I let the pointer of the last node point to NULL since it doesn't have any node to point to, and the head will be equal to p.

```
list* push_front(list* head, list* personToAdd)
{
   list* p = (list*) malloc (sizeof(list));
   strcpy(p -> n.f_name, personToAdd -> n.f_name);
   strcpy(p -> n.s_name, personToAdd -> n.s_name);
   strcpy(p -> w.f_name, personToAdd -> w.f_name);
   strcpy(p -> w.s_name, personToAdd -> w.s_name);
   strcpy(p -> category, personToAdd -> category);
   strcpy(p -> type, personToAdd -> type);
   p -> d.year = personToAdd -> d.year;
   p -> d.month = personToAdd -> d.month;
   p -> d.day = personToAdd -> d.day;

   p -> nxt = head;
   head = p;
   return head;
}
```

9. **printPerson()** is a void function that receives a list, I used this function for printing all the information of the node of the list.

```
void printPerson(list* p)
{
   if (p == NULL)
   {
      printf("head is NULL\n");
      return;
   }
   if (strcmp(p->w.f_name, "None")==0)
   {
      printf("%s %s | %s | %s | %d/%d/%d\n", p->n.f_name, p->n.s_name, p->type, p->category, p->d.day, p->d.month, p->d.year);
   }
   else
   {
      printf("%s %s and %s %s | %s | %s | %d/%d/%d\n", p->n.f_name, p->n.s_name, p->w.f_name, p->w.s_name, p->type, p->category, p->d.day, p-
>d.month, p->d.year);
   }
}
```

10. **data_search()** is a void function that receives a list, I used this function for the users to choose the options which are displayed on the search menu that will be shown when the users choose the searching for the data in the databases option in the databases menu. In this function, I implemented the same switch case in the conditional statement method just like the other options functions.

```
void data_search(list* head)
{
   int ops = 0;
   while(ops!=4)
   {
      system("cls");
      printf("_____ __\n");
      printf("__ ___/__ ____/__   |__ __ \\_ ____/__ / / /\n");
      printf("____ \\__ __/  __ /| |_ /_/ / /   __ /_/ / \n");
      printf("____/ /_ /___ _ ___ |  _, _/// /___ _ __  /  \n");
      printf("/____/ /_____/ /_/ |_/_/ |_| \\____/ /_/ /_/  \n");
      printf("\n  1. Search by Category\n");
      printf("  2. Search by Day\n");
      printf("  3. The events that occur in the last month and those that will occur in the next month\n");
      printf("  4. Back\n");
      scanf("%d", &ops);
      switch(ops)
      {
```

```
        case 1:
          search_category(head);
          break;
        case 2:
          search_day(head);
          break;
        case 3:
          last_next(head);
          break;
        default:
          printf("Please choose the available options\n");
      }
    }
}
```

11. **search_category()** is a void function that receives a list, I used this function for the users to choose the category of data that they want to search for in the database. In this function, I implement the switch case to let the user choose which category the user wants to display, and I declare a list pointer type variable called p, initialize it equal to head which is the first node. Then, I make a loop and give a condition to compare the string str with the category inside the p and if it is the same then I used the printPerson() function to print the node. After that, I moved to the next node, and this keeps on through every node until it goes through all the nodes.

```
void search_category(list* head)
{
  system("cls");
  printf("_____ __\n");
  printf("__ ____/__   |__ __/__ ____/_ ____/_ __ \\__ __ \\ \\/ /\n");
  printf("_ /    __ /| |_ /  __ __/  _ / __ _ / / /_ /_/ /_  / \n");
  printf("/ /___ _ ___| /  _ /___ / /_/ / / / /_/ /_  _, _/_  /  \n");
  printf("\\____/ /_/ |_/_/   /_____/ \\____/ \\____/ /_/ |_| /_/   \n");
  printf("\nSelect the Category of the users that you want to display:\n");
  int ctr=5;

  char str[100];
  while(ctr>4 || ctr<1)
  {
    printf("Category of the users:\n");
    printf("1. Friend\n");
    printf("2. Family\n");
    printf("3. Relative\n");
    printf("4. Colleague\n");
    scanf("%d", &ctr);
    switch(ctr)
    {
      case 1:
        strcpy(str, "Friend");
        break;
      case 2:
        strcpy(str, "Family");
        break;
      case 3:
        strcpy(str, "Relative");
        break;
      case 4:
        strcpy(str, "Colleague");
        break;
```

```
            default:
                printf("Please choose the available options\n");
                break;
        }
    }
    system("cls");
    printf("Result of the searching information:\n");
    list* p = head;
    while (p != NULL) {
        if (strcmp(p->category, str) == 0) {
            printPerson(p);
        }
        p = p -> nxt;
    }

    //hold the output
    int b = 0;
    while (b != 4) {
        printf("\n\nEnter 4 to go back\n");
        scanf("%d", &b);
    }
}
```

12. **search_day()** is a void function that receives a list, I used this function for the users to choose the type of day of the data that they want to search for in the database. In this function, I did the same way as the search_category() function but I changed the options to choose the type of day instead.

```
void search_day(list* head)
{
    system("cls");
    printf("_____ __\n");
    printf("___ __ \\__  | \\/ /\n");
    printf("__ / / /_ /| |_ / \n");
    printf("_ /_/ /_ ___| / \n");
    printf("/____/ /_/ |_/_/  \n");
    printf("\nSelect the Type of day that you want to display:\n");
    int td=5;
    char str[100];
    while(td>4 || td<1)
    {
        printf("Type of day:\n");
        printf("1. Birthday\n");
        printf("2. Anniversary\n");
        printf("3. Nameday\n");
        printf("4. Marriage\n");
        scanf("%d", &td);

        switch(td)
        {
            case 1:
                strcpy(str, "Birthday");
                break;
            case 2:
```

```
            strcpy(str, "Anniversary");
            break;
        case 3:
            strcpy(str, "Nameday");
            break;
        case 4:
            strcpy(str, "Marriage");
            break;
        default:
            printf("Please choose the available options\n");
            break;
      }
    }
    system("cls");
    printf("Result of the searching information:\n");
    list* p = head;
    while (p != NULL) {
      if (strcmp(p->type, str) == 0) {
        printPerson(p);
      }
      p = p -> nxt;
    }

    //hold the output
    int b = 0;
    while (b != 4) {
      printf("\n\nEnter 4 to go back\n");
      scanf("%d", &b);
    }
}
```

13. **last_next()** is a void function that takes a list, I used this function for displaying the day that occurred last month and the day that will occur next month. In this function, I check the month by using the time.h library to use the time_h data type, time() function, struct tm structure, and *localtime() function for extracting the real month from the computer and use it for checking to select the day that occurred last month and next month. Since the months in a year have 12, for the last month, I decided to create an if-else statement to check if the month is January then the last month should be December besides that just decrement the month by 1 then the program will display the day that occurred last month and for next month, I decided to create another if-else statement to check if the month is December then the next month should be January and besides that just increment the month by 1 then the program will display the day that will occur next month. After that, I declare a list pointer type called p and let it point to the address of the head which is the first node that stores the information of the first person in the database. Then, I make a loop to and give the if conditional statement if the month data inside that node is equal to the month1 which is the last month then print that node using the printPerson(p) function with the address of p that I already declared, then p will go to the next node and check the month of the next node and print it again if the month is last month. This loop will keep on until it goes through all the nodes. The same goes for the next month, I also used the same method for checking and printing. To make the data print out in chronological order, I declare two temporary lists for storing the list of last month's and next month's nodes, and I used the Bubble sorting algorithm by creating another function to call in this function called bubbleSort() to sort those two lists in chronological order then print the list out. In the end, I free both temporary lists after displaying all the data inside the list on the screen.

```
void last_next(list* head)
{
    system("cls");
    time_t t = time(NULL);
```

Name: Mean Diamand
Neptun Code: KSG25Z

```
struct tm tm = *localtime(&t);
int month1=tm.tm_mon+1;
if(month1==1)
{
   month1=12;
}
else
{
   month1-=1;
}
printf("The events that occurred last month:\n\n");
list *lt= NULL;
list* p = head;
while(p!=NULL)
{
   if(p->d.month==month1)
   {
      lt = insert_data(lt, p);
   }
   p=p->nxt;
}
bubbleSort(lt);
list* i = lt;
while(i!=NULL)
{
   printPerson(i);
   i=i->nxt;
}

int month2=tm.tm_mon+1;
printf("\n\nThe events that will occur next month:\n\n");
if(month2==12)
{
   month2=1;
}
else
{
   month2+=1;
}
p=head;
list *nt= NULL;
while(p!=NULL)
{
   if(p->d.month==month2)
   {
      nt = insert_data(nt, p);
   }
   p=p->nxt;
}
bubbleSort(nt);
i = nt;
while(i!=NULL)
{
```

```
    printPerson(i);
    i=i->nxt;
  }

  // hold the output
  int b = 0;
  while (b != 4)
  {
    printf("\n\nEnter 4 to go back\n");
    scanf("%d", &b);
  }
  free(lt);
  free(nt);
}
```

14. **bubbleSort()** is a void function that receives a list, I used this function for implementing the Bubble sorting algorithm on the list. First, I declare a flag variable called "swapped" and two lists. I checked the received list if it is empty then return nothing, if not then it will continue into the do while loop in this loop, swapped variable will be equal to 0 and ptr1 will be equal to start then we will go into the while loop to check if the day of the node is greater than the day of the next node or not. If yes then swap() function will be called to swap these two values and swapped variable will become 1. ptr1 will move to the next node and we will keep checking until there is no swapping left.

```
void bubbleSort(list *start)
{
  int swapped;
  list *ptr1;
  list *lptr = NULL;

  /* Checking for empty list */
  if (start == NULL)
    return;

  do
  {
    swapped = 0;
    ptr1 = start;

    while (ptr1->nxt != lptr)
    {
      if ((ptr1->d.day) > (ptr1->nxt->d.day))
      {
        swap(ptr1, ptr1->nxt);
        swapped = 1;
      }
      ptr1 = ptr1->nxt;
    }
    lptr = ptr1;
  }
  while (swapped);
}
```

Name: Mean Diamand
Neptun Code: KSG25Z

15. **swap()** is a void function that receives two addresses of the data of the node and the data of the next node, I used this function for swapping the value of the two nodes that are next to each other by using the famous swapping method by declaring a temporary node called "temp" and allocate space for that node, then let temp equal to a, a equal to b and b equal to temp. After that, I will free that temporary node.

```
void swap(list *a, list *b)
{
   list* temp = (list*) malloc (sizeof(list));
   // temp = a
   strcpy(temp->n.f_name, a->n.f_name);
   strcpy(temp->n.s_name, a->n.s_name);
   strcpy(temp->w.f_name, a->w.f_name);
   strcpy(temp->w.s_name, a->w.s_name);
   strcpy(temp->category, a->category);
   strcpy(temp->type, a->type);
   temp->d.year = a->d.year;
   temp->d.month = a->d.month;
   temp->d.day = a->d.day;

   // a = b
   strcpy(a->n.f_name, b->n.f_name);
   strcpy(a->n.s_name, b->n.s_name);
   strcpy(a->w.f_name, b->w.f_name);
   strcpy(a->w.s_name, b->w.s_name);
   strcpy(a->category, b->category);
   strcpy(a->type, b->type);
   a->d.year = b->d.year;
   a->d.month = b->d.month;
   a->d.day = b->d.day;

   // b = temp
   strcpy(b->n.f_name, temp->n.f_name);
   strcpy(b->n.s_name, temp->n.s_name);
   strcpy(b->w.f_name, temp->w.f_name);
   strcpy(b->w.s_name, temp->w.s_name);
   strcpy(b->category, temp->category);
   strcpy(b->type, temp->type);
   b->d.year = temp->d.year;
   b->d.month = temp->d.month;
   b->d.day = temp->d.day;

   free(temp);
}
```

16. **data_delete()** is a function that receives a list and an address of an integer parameter, I used this function for the user to input the first name and surname of the person that they wanted to delete, but before letting them input the first name and surname, I made a while loop and call the printPerson() function to print all the data of the nodes inside the list to display it for the users to check which person that they want to delete from the databases. After inputting the first name and surname, all the data will be displayed once again after deleting the data to solve that the data of the input person has been deleted. In the end, the new modified list will be returned.

```
list *data_delete(list* head, int *numOfPeople)
{
```

```
    system("cls");
    printf("_____\n");
    printf("___  __ \\__  ____/__  /___  ____/__  __/__  ____/\n");
    printf("__  / / /_  __/  __  / __  __/  __  / __  __/   \n");
    printf("_  /_/ /_  /___  _  /___  /___  _  /   _  /___   \n");
    printf("/_____/ /_____/  /_____/_____/  /_/   /_____/   \n\n");
    list* p = head;
    while (p != NULL)
    {
        printPerson(p);
        p = p->nxt;
    }
    char f_name[20];
    char s_name[20];
    printf("\nInput the name of the person that you want to delete:\n");
    printf("Name(Firstname Surname): ");
    scanf("%s %s", f_name, s_name);
    head=delete_person(head, f_name, s_name);
    (*numOfPeople)--;
    printf("\nResult after deleting the data:\n");
    p=head;
    while(p!=NULL)
    {
        printPerson(p);
        p=p->nxt;
    }

    // hold the output
    int b = 0;
    while (b != 4)
    {
        printf("\n\nEnter 4 to go back\n");
        scanf("%d", &b);
    }
    return head;
}
```

17. **delete_person()** is a function that receives a list, and two character type addresses. I used it for deleting the given node from the list. In this function, I checked if the list is empty or not, if it is empty then return NULL. If not, then it will continue to check by using strcmp() function to compare the first name and surname of the first node. If it has the same name as the person that we want to delete then we will call the pop_front() function for deleting the first node. If not, then we will keep iterating through the list to find the node that we want to delete and then return the list.

```
list *delete_person(list *head, char* f, char* s)
{
    list *p = head;
    if(head == NULL)
    {
        return head;
    }
    if(strcmp(head->n.f_name, f)==0 && strcmp(head->n.s_name, s)==0)
    {
        return pop_front(head);
    }
```

```
    while(p->nxt != NULL && (strcmp(p->nxt->n.f_name, f)!=0 && strcmp(head->n.s_name, s))!=0)
    {
        p=p->nxt;
    }
    if(p->nxt!=NULL)
    {
        list *q=p->nxt;
        p->nxt=q->nxt;
        free(q);
    }
    return head;
}
```

18. **pop_front()** is a function that receives a list and returns the list. I used this function to delete the first node of the list.

```
list *pop_front(list *head)
{
    if(head!=NULL)
    {
        list *p=head;
        head=head->nxt;
        free(p);
    }
    return head;
}
```

19. **print_databases()** is a function that receives a list. I used this function to print the list and all the information inside each node by calling the printPerson() function.

```
void print_databases(list *head)
{
    system("cls");
    printf("_____\n");
    printf("___ __ \\__ |__ __/__ |__ __)__ |_ ___/__ ____/_ ___/\n");
    printf("__ / / /_ /| |_ / __ /| |_ __ |_ /| |___ \\__ __/ _____ \\ \n");
    printf("_ /_/ /_ ___| / _ ___| /_/ /_ ___ |___/ /_ /___ ____/ / \n");
    printf("/_____/ /_/ |_/_/ /_/ |_/_____/ /_/ |_/____/ /_____/ /____/ \n\n");
    list* p = head;
    while (p != NULL)
    {
        printPerson(p);
        p = p->nxt;
    }

    // hold the output
    int b = 0;
    while (b != 4)
    {
        printf("\n\nEnter 4 to go back\n");
        scanf("%d", &b);
    }
}
```

Name: Mean Diamand
Neptun Code: KSG25Z

In the main function, I created a loop that will keep asking the users for their choice of options, this choice will be from asked from the introduction() function that has been called in the loop. After the user input the choice, the program will run into the switch case and jump to the statement according to the choice. I also call the read_the_file() function to read the text file for displaying on the console.
Notice: In some functions like data_entry(), search_category(), search_day(), and last_next(), I did the loop that help to hold the output. For example something like this below(from the search_category() function) :

```
int b = 0;
   while (b != 4)
   {
      printf("\n\nEnter 4 to go back\n");
      scanf("%d", &b);
   }
```

I did this to prevent the function, to jump back by itself to where it is called.