Database For Student Using Linked List Data Structure :

- linkedList.h file

linked_list.h

```
/* linked_list.h
 */

#ifndef LINKED_LIST_H_
#define LINKED_LIST_H_
#include "stdio.h"
#include "stdlib.h"
#include "string.h"
#include "conio.h"

#define DPRINTF(...)    {fflush(stdout);\
        fflush(stdin);\
        printf(__VA_ARGS__);\
        fflush(stdout);\
        fflush(stdin);}
//effective data
struct Sdata
{
    int ID;
    char name[40];
    float height ;

};

//linked list node
struct SStudent
{
    struct Sdata student;
    struct SStudent* PNextStudent ;
};

struct SStudent* gpFirstStudent;


void AddStudent();
int delete_student ();
int view_node();
void ReverseList();
void lengthOfLinkedList();
int Recursion_lengthOfLinkedList(struct SStudent *pSelectedStudent);
void view_students();
void DeleteAll();
void middleList();
int viewNodeFromEnd();

#endif /* LINKED_LIST_H_ */
```

- main.c File

```c
/* main.c */

#include "linked_list.h"

int main()
{
    char temp_text[40];
    int count;
    while(1)
    {
        DPRINTF("\n ==================");
        DPRINTF("\n\t Choose on of the Following Option :\n");
        DPRINTF("\n 1:AddStudent ");
        DPRINTF("\n 2:DeleteStudent ");
        DPRINTF("\n 3:viewStudent ");
        DPRINTF("\n 4: DeleteAll");
        DPRINTF("\n 5: viewNode ");
        DPRINTF("\n 6: lengthOfLinkedList ");
        DPRINTF("\n 7: Recursion_lengthOfLinkedList ");
        DPRINTF("\n 8: ReverseList ");
        DPRINTF("\n 9: middleList ");
        DPRINTF("\n 10: viewNodeFromEnd ");
        DPRINTF("\n Enter Option Number :");
        gets(temp_text);
        DPRINTF("\n ====================");
        DPRINTF("\n ==================== );
        switch(atoi(temp_text))
        {
        case 1:
            AddStudent();
            break;
        case 2:
            delete_student();
            break;
        case 3:
            view_students();
            break;
        case 4:
            DeleteAll();
            break;
        case 5:
            view_node();
            break;
        case 6:
            lengthOfLinkedList();
            break;
        case 7:
            count=Recursion_lengthOfLinkedList(gpFirstStudent);
            DPRINTF("\nThe Length of Linked List = %d",count);
            break;
        case 8:
            ReverseList();
            break;
        case 9:
            middleList();
            break;
        case 10:
            viewNodeFromEnd();
            break;
```

- LinkedList.c File

Add Student Function :

```
void AddStudent()
{
    char temp_text[50];
    struct SStudent* pNewStudent ;
    struct SStudent* pLastStudent ;
    //check list is empty ==yes
    if(gpFirstStudent == NULL)
    {
        pNewStudent=(struct SStudent*) malloc (sizeof(struct SStudent));
        //assign it to gpfirst
        gpFirstStudent=pNewStudent;
    }else //list Contains Records
    {
        pLastStudent=gpFirstStudent;
        while (pLastStudent->PNextStudent)
            pLastStudent=pLastStudent->PNextStudent;
        pNewStudent=(struct SStudent*) malloc(sizeof(struct SStudent));
        pLastStudent->PNextStudent=pNewStudent;
    }
    //fill new Record
    DPRINTF("\n Enter The ID :");
    gets(temp_text);
    pNewStudent->student.ID=atoi(temp_text);

    DPRINTF("\n Enter Student Full Name :");
    gets(pNewStudent->student.name);

    DPRINTF("\n Enter Student Height :");
    gets(temp_text);
    pNewStudent->student.height=atoi(temp_text);

    //Set The Next Pointer (New_Student) NULL
    pNewStudent->PNextStudent=NULL;
```

Delete Student Function :

```
int delete_student ()
{
    char temp_text[40];
    unsigned int selected_id;
    //get the The Selected id
    DPRINTF("\n Enter The Student id to Be Deleted :");
    gets(temp_text);
    selected_id=atoi(temp_text);

    //List is not Empty
    if (gpFirstStudent)
    {
        struct SStudent *pSelectedStudent=gpFirstStudent;
        struct SStudent *pPreviousStudent=NULL;

        //loop on all Records
        while(pSelectedStudent)
        {
            //Compare each Node with The Selected ID
            if(pSelectedStudent->student.ID ==selected_id)
            {
                if(pPreviousStudent)//The first is not The Selected
                {
                    pPreviousStudent->PNextStudent=pSelectedStudent->PNextStudent;
                }else{ //1st Student == ID
                    gpFirstStudent=pSelectedStudent->PNextStudent;
                }
                free(pSelectedStudent);
                return 1;
            }
            pPreviousStudent=pSelectedStudent;
            pSelectedStudent=pSelectedStudent->PNextStudent;
        }
    }
```

View Node Function :

```
int view_node()
{
    unsigned int SelectedIndex,count=0;
    char temp_text[50];
    DPRINTF("\nEnter Node Index :");
    gets(temp_text);
    SelectedIndex=atoi(temp_text);

    //To Check if The List is Empty of NOt
    if(gpFirstStudent)
    {
        struct SStudent *pSelectedStudent=gpFirstStudent;
        while(pSelectedStudent)
        {
            if (count==SelectedIndex)
            {
                DPRINTF("\nThe Information of Student with has Index %d :",count);
                DPRINTF("\nStudent ID : %d",pSelectedStudent->student.ID);
                DPRINTF("\nStudent Name : %s",pSelectedStudent->student.name);
                DPRINTF("\nStudent Height : %f",pSelectedStudent->student.height);
                return 1;
            }
            pSelectedStudent=pSelectedStudent->PNextStudent;
            count ++;
        }
        DPRINTF("The Index not Exist \n");
    }
    else
    {
        DPRINTF("Empty List ");
        return 0;
    }
```

Reverse List Function :

```
}

void ReverseList()
{
    struct SStudent *pPreviousStudent=NULL;
    struct SStudent *pCurrentStudent=gpFirstStudent;
    struct SStudent *pNextStudent=NULL;

    while (pCurrentStudent)
    {
        pNextStudent=pCurrentStudent->PNextStudent;
        pCurrentStudent->PNextStudent=pPreviousStudent;
        pPreviousStudent=pCurrentStudent;
        pCurrentStudent=pNextStudent;
    }
    gpFirstStudent=pPreviousStudent;

}
```

Length of LinkedList Function :

```
void lengthOfLinkedList()
{
    unsigned int count =0;
    if(gpFirstStudent)
    {
        struct SStudent *pSelectedStudent=gpFirstStudent;
        while(pSelectedStudent)
        {
            pSelectedStudent=pSelectedStudent->PNextStudent;
            count ++;
        }
        DPRINTF("\nThe Length of Linked List = %d",count);

    }
    else
    {
        DPRINTF("\nThe Length of Linked List = %d ",count);

    }

}
```

Length of Linked List Using Recursion method Function :

```
int Recursion_lengthOfLinkedList(struct SStudent *pSelectedStudent)
{
    if(pSelectedStudent)
    {
        pSelectedStudent=pSelectedStudent->PNextStudent;
        return (1+Recursion_lengthOfLinkedList(pSelectedStudent));
    }
    else
        return 0;

}
```

View Students Function :

```
void view_students()
{
    struct SStudent* pCurrentStudent =gpFirstStudent;
    int count =0;
    if(pCurrentStudent==NULL)
    {
        DPRINTF("\n Empty List \n");
    }
    else
    {
        while(pCurrentStudent)
        {
            DPRINTF("\n Record Number %d",count+1);
            DPRINTF("\n\t ID: %d",pCurrentStudent->student.ID);
            DPRINTF("\n\t Name: %s",pCurrentStudent->student.name);
            DPRINTF("\n\t Height: %f",pCurrentStudent->student.height);
            pCurrentStudent=pCurrentStudent->PNextStudent;
            count++;
        }
    }
}
```

Delete All Function :

```
void DeleteAll()
{
    struct SStudent* pCurrentStudent =gpFirstStudent;
    if(pCurrentStudent==NULL)
    {
        DPRINTF("\n Empty List \n");
    }
    else
    {
        while(pCurrentStudent)
        {
            struct SStudent* pTempStudent =pCurrentStudent;
            pCurrentStudent=pCurrentStudent->PNextStudent;
            free(pTempStudent);
        }
        gpFirstStudent=NULL;
    }
}
```

The Index of The Middle of List Function :

```
void middleList()
{
    unsigned int length ,middle;
    struct SStudent* pCurrentStudent =gpFirstStudent;
    if(pCurrentStudent==NULL)
    {
        DPRINTF("\n Empty List \n");
    }
    length =Recursion_lengthOfLinkedList(pCurrentStudent);
    middle =(length /2);

    DPRINTF("\n The middle =%d",middle);

}
```

View Node From End Function :

```c
int viewNodeFromEnd()
{

    unsigned int NodeNumberFromEnd,length=0,NodeNumberFrombegin=0,count=0;

    char temp_text[50];
    DPRINTF("\nEnter Node Number :");
    gets(temp_text);
    NodeNumberFromEnd=atoi(temp_text);

    length =Recursion_lengthOfLinkedList(gpFirstStudent);

    NodeNumberFrombegin=length-NodeNumberFromEnd-1;



    //To Check if The List is Empty of NOt
    if(gpFirstStudent)
    {
        struct SStudent *pSelectedStudent=gpFirstStudent;
        while(pSelectedStudent)
        {
            if (count==NodeNumberFrombegin)
            {
                DPRINTF("\nThe Information of Student with has Index %d :",count);
                DPRINTF("\nStudent ID : %d",pSelectedStudent->student.ID);
                DPRINTF("\nStudent Name : %s",pSelectedStudent->student.name);
                DPRINTF("\nStudent Height : %f",pSelectedStudent->student.height);
                return 1;
            }
            pSelectedStudent=pSelectedStudent->PNextStudent;
            count ++;
        }
        DPRINTF("The Index not Exist \n");
```

Output of The LinkedList Program :

-add Student :

```
==================
        Choose on of the Following Option :

1:AddStudent
2:DeleteStudent
3:viewStudent
4: DeleteAll
5: viewNode
6: lengthOfLinkedList
7: Recursion_lengthOfLinkedList
8: ReverseList
9: middleList
10: viewNodeFromEnd
Enter Option Number :1

==================
Enter The ID :1

Enter Student Full Name :Ephraim

Enter Student Height :175
```

-add another Student :

```
==================
        Choose on of the Following Option :

1:AddStudent
2:DeleteStudent
3:viewStudent
4: DeleteAll
5: viewNode
6: lengthOfLinkedList
7: Recursion_lengthOfLinkedList
8: ReverseList
9: middleList
10: viewNodeFromEnd
Enter Option Number :1

====================
Enter The ID :2

Enter Student Full Name :Anton

Enter Student Height :180
```

-add another Student :

```
==================
        Choose on of the Following Option :

1:AddStudent
2:DeleteStudent
3:viewStudent
4: DeleteAll
5: viewNode
6: lengthOfLinkedList
7: Recursion_lengthOfLinkedList
8: ReverseList
9: middleList
10: viewNodeFromEnd
Enter Option Number :1

====================
Enter The ID :3

Enter Student Full Name :Ayman

Enter Student Height :170
```

-View Students DataBase

```
------------------
        Choose on of the Following Option :

1:AddStudent
2:DeleteStudent
3:viewStudent <--
4: DeleteAll
5: viewNode
6: lengthOfLinkedList
7: Recursion_lengthOfLinkedList
8: ReverseList
9: middleList
10: viewNodeFromEnd
Enter Option Number :3 <--

====================
Record Number 1
        ID: 1
        Name: Ephraim
        Height: 175.000000
Record Number 2
        ID: 2
        Name: Anton
        Height: 180.000000
Record Number 3
        ID: 3
        Name: Ayman
        Height: 170.000000
------------------
```

-Use Function Length of LinkedList :

```
==================
        Choose on of the Following Option :

1:AddStudent
2:DeleteStudent
3:viewStudent
4: DeleteAll
5: viewNode
6: lengthOfLinkedList
7: Recursion_lengthOfLinkedList
8: ReverseList
9: middleList
10: viewNodeFromEnd
Enter Option Number :6

==================
The Length of Linked List = 3
==================
```

-Use Function Length of LinkedList by Recursion :

```
        Choose on of the Following Option :

1:AddStudent
2:DeleteStudent
3:viewStudent
4: DeleteAll
5: viewNode
6: lengthOfLinkedList
7: Recursion_lengthOfLinkedList
8: ReverseList
9: middleList
10: viewNodeFromEnd
Enter Option Number :7

==================
The Length of Linked List = 3
==================
```

-Use Function The middle Index of The Linked List :

```
        Choose on of the Following Option :

1:AddStudent
2:DeleteStudent
3:viewStudent
4: DeleteAll
5: viewNode
6: lengthOfLinkedList
7: Recursion_lengthOfLinkedList
8: ReverseList
9: middleList
10: viewNodeFromEnd
Enter Option Number :9

==================
The middle =1
------------------
```

-Check The Function “view The Node From End” :

```
==================
        Choose on of the Following Option :

1:AddStudent
2:DeleteStudent
3:viewStudent
4: DeleteAll
5: viewNode
6: lengthOfLinkedList
7: Recursion_lengthOfLinkedList
8: ReverseList
9: middleList
10: viewNodeFromEnd
Enter Option Number :10

==================
Enter Node Number :0

The Information of Student with has Index 2 :
Student ID : 3
Student Name : Ayman
Student Height : 170.000000
==================
```

-Check The Function “Reverse The LinkedList “ :

```
==================
        Choose on of the Following Option :

1:AddStudent
2:DeleteStudent
3:viewStudent
4: DeleteAll
5: viewNode
6: lengthOfLinkedList
7: Recursion_lengthOfLinkedList
8: ReverseList
9: middleList
10: viewNodeFromEnd
Enter Option Number :8

--------------------
==================
        Choose on of the Following Option :

1:AddStudent
2:DeleteStudent
3:viewStudent
4: DeleteAll
5: viewNode
6: lengthOfLinkedList
7: Recursion_lengthOfLinkedList
8: ReverseList
9: middleList
10: viewNodeFromEnd
Enter Option Number :3

==================
Record Number 1
        ID: 3
        Name: Ayman
        Height: 170.000000
Record Number 2
        ID: 2
        Name: Anton
        Height: 180.000000
Record Number 3
        ID: 1
        Name: Ephraim
        Height: 175.000000
------------------
```

view Students
After Reversing

-Check The Function “DeleteAll” :

```
        Choose on of the Following Option :

1:AddStudent
2:DeleteStudent
3:viewStudent
4: DeleteAll
5: viewNode
6: lengthOfLinkedList
7: Recursion_lengthOfLinkedList
8: ReverseList
9: middleList
10: viewNodeFromEnd
Enter Option Number :4

====================
====================
        Choose on of the Following Option :

1:AddStudent
2:DeleteStudent
3:viewStudent
4: DeleteAll
5: viewNode
6: lengthOfLinkedList
7: Recursion_lengthOfLinkedList
8: ReverseList
9: middleList
10: viewNodeFromEnd
Enter Option Number :3

====================
Empty List
```

View Student After Use Function Delete_All