

Unit 4 Lesson 1

Guirguis Hedia

FIFO DATA Structure :

- main.c File

```
2+ | * main.c
7
8 #include "fifo.h"
9
10
11 void main()
12 {
13     FIFO_Buf_t FIFO_UART;
14     element_type i,temp;
15     if(FIFO_init(&FIFO_UART,uart_buffer,5)==FIFO_no_error)
16         printf("FIFO init ---Done \n");
17
18     for (i=0;i<7;i++)
19     {
20         printf("FIFO Enqueue (%x) \n",i);
21         if(FIFO_enqueue(&FIFO_UART,i)==FIFO_no_error)
22             printf("FIFO enqueue -----Done \n");
23         else
24             printf("FIFO enqueue ----- Failed\n");
25     }
26
27     FIFO_print(&FIFO_UART);
28
29
30     if(FIFO_dequeue(&FIFO_UART,&temp)==FIFO_no_error)
31         printf("FIFO dequeue %x -----Done \n",temp);
32
33     if(FIFO_dequeue(&FIFO_UART,&temp)==FIFO_no_error)
34         printf("FIFO dequeue %x -----Done \n",temp);
35
36     FIFO_print(&FIFO_UART);
37
38 }
39
```

- FIFO.h File

```

20 | * fifo.h
7
8 #ifndef FIFO_H_
9 #define FIFO_H_
10
11 #include "stdio.h"
12 #include "stdint.h"
13
14 //USER Configuration
15 //Select The Element Type (uint8_t , uint16_t , uint32_t , ....)
16 #define element_type uint8_t
17
18 //Create buffer
19 #define width 5
20 element_type uart_buffer [width];
21
22 //FIFO Data Types
23 typedef struct {
24     unsigned int length;
25     unsigned int count ;
26     element_type* head ;
27     element_type* tail ;
28     element_type* base ;
29 }FIFO_Buf_t;
30
31 typedef enum{
32     FIFO_no_error,
33     FIFO_full,
34     FIFO_empty,
35     FIFO_null
36 }FIFO_Buf_Status;
37
38
39
40 //FIFO APIs
41 FIFO_Buf_Status FIFO_init(FIFO_Buf_t *fifo,element_type* buf,uint32_t length);
42 FIFO_Buf_Status FIFO_enqueue (FIFO_Buf_t *fifo ,element_type item);
43 FIFO_Buf_Status FIFO_dequeue (FIFO_Buf_t *fifo ,element_type* item);
44 FIFO_Buf_Status FIFO_IS_FULL (FIFO_Buf_t *fifo );
45 void FIFO_print (FIFO_Buf_t *fifo );
46
47 #endif /* FIFO_H_ */
48

```

Problems Tasks Console Properties Debugger Console

- FIFO.c File

FIFO_init() Function And FIFO_enqueue() Function :

```

9
10
11 FIFO_Buf_Status FIFO_init(FIFO_Buf_t *fifo, element_type* buf, uint32_t length)
12 {
13     if(buf==NULL)
14         return FIFO_null;
15
16     fifo->base =buf;
17     fifo->head =buf;
18     fifo->tail =buf;
19     fifo->length =length;
20     fifo->count =0;
21
22     return FIFO_no_error;
23 }
24
25 FIFO_Buf_Status FIFO_enqueue (FIFO_Buf_t *fifo ,element_type item)
26 {
27     if(!fifo->base || !fifo->head || !fifo->tail)
28         return FIFO_null;
29
30     if(FIFO_IS_FULL(fifo)==FIFO_full)
31         return FIFO_full;
32
33     *(fifo->head)=item;
34     fifo->count++;
35
36     //circular FIFO
37     if(fifo->head ==(fifo->base +(fifo->length *sizeof(element_type))))
38         fifo->head=fifo->base;
39     fifo->head++;
40
41     return FIFO_no_error;
42 }

```

FIFO_dequeue() Function And FIFO_is_FULL() Function :

```

}
FIFO_Buf_Status FIFO_dequeue (FIFO_Buf_t *fifo ,element_type* item)
{
    if(!fifo->base || !fifo->head || !fifo->tail)
        return FIFO_null;

    //Check if FIFO is Empty
    if(fifo->count ==0)
        return FIFO_empty;

    *item=*(fifo->tail);
    fifo->count--;

    if(fifo->tail ==(fifo->base +(fifo->length *sizeof(element_type))))
        fifo->tail=fifo->base;
    else
        fifo->tail++;

    return FIFO_no_error;
}

FIFO_Buf_Status FIFO_IS_FULL (FIFO_Buf_t *fifo )
{
    if(!fifo->base || !fifo->head || !fifo->tail)
        return FIFO_null;
    if(fifo->count == fifo->length)
        return FIFO_full;

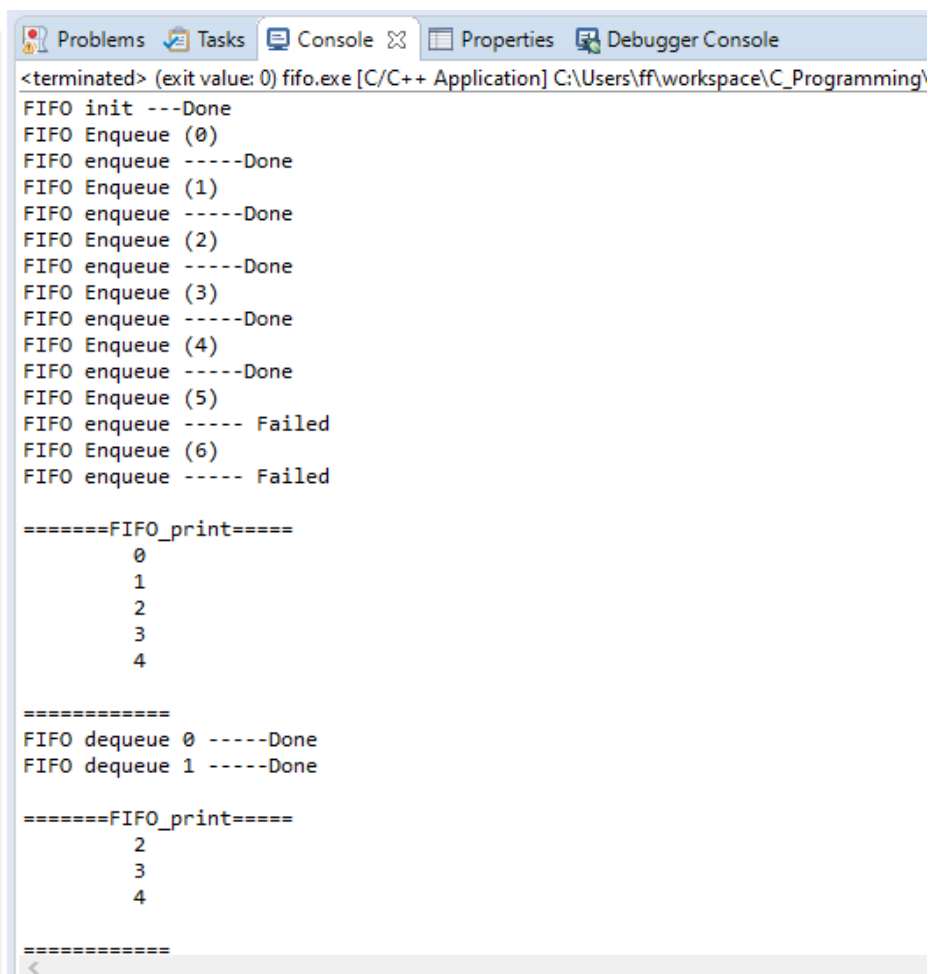
    return FIFO_no_error;
}

```

FIFO_print() Function :

```
72 }
73 void FIFO_print (FIFO_Buf_t *fifo )
74 {
75     int i;
76     element_type* temp;
77
78     if(fifo->count ==0)
79     {
80         printf("fifo is empty \n");
81     }else{
82         temp =fifo->tail;
83         printf("\n=====FIFO_print=====\n");
84         for(i=0;i<fifo->count;i++)
85         {
86             printf("\t %x\n",*temp);
87             temp++;
88         }
89         printf("\n=====\\n");
90     }
91 }
92
```

FIFO Output :



```
<terminated> (exit value: 0) fifo.exe [C/C++ Application] C:\Users\ff\workspace\C_Programming\
FIFO init ---Done
FIFO Enqueue (0)
FIFO enqueue -----Done
FIFO Enqueue (1)
FIFO enqueue -----Done
FIFO Enqueue (2)
FIFO enqueue -----Done
FIFO Enqueue (3)
FIFO enqueue -----Done
FIFO Enqueue (4)
FIFO enqueue -----Done
FIFO Enqueue (5)
FIFO enqueue ----- Failed
FIFO Enqueue (6)
FIFO enqueue ----- Failed

=====FIFO_print=====
    0
    1
    2
    3
    4

=====
FIFO dequeue 0 -----Done
FIFO dequeue 1 -----Done

=====FIFO_print=====
    2
    3
    4

=====
```

FIFO DATA Structure :

- main.c File

```
7
8 #include "lifo.h"
9
10 int main()
11 {
12     element_type i,temp=0 ;
13
14     LIFO_Buf_t uart_lifo ,I2C_lifo;
15     //static allocation
16     LIFO_init(&uart_lifo,buffer,5);
17
18     //dynamic allocation
19     element_type* buffer2=(element_type*)malloc(5 *sizeof(unsigned int));
20     LIFO_init(&I2C_lifo,buffer2,5);
21     printf("=====LIFO ADD=====\\n");
22     for (i=0 ;i<8;i++)
23     {
24         if(LIFO_Add_item(&uart_lifo,i)==LIFO_no_error)
25             printf("UART_LIFO add: %d \\n",i);
26     }
27     printf("=====LIFO GET=====\\n");
28     for (i=0 ;i<8;i++)
29     {
30         if(LIFO_get_item(&uart_lifo,&temp)==LIFO_no_error)
31             printf("UART_LIFO get: %d \\n",temp);
32     }
33
34     return 0;
35 }
36
37
```

- LIFO.h file

```
20 * lifo.h
7
8 #ifndef LIFO_H_
9 #define LIFO_H_
10 #include "stdlib.h"
11 #include "stdint.h"
12
13 //type Definitions
14 #define element_type    uint8_t
15 #define width 5
16 element_type buffer[width];
17
18
19 //Select The Element Type (uint8_t , uint16_t , uint32_t,.....)
20
21 typedef struct {
22     unsigned int length ;
23     unsigned int count ;
24     element_type *base ;
25     element_type* head ;
26 }LIFO_Buf_t;
27
28 typedef enum{
29     LIFO_no_error,
30     LIFO_full,
31     LIFO_empty,
32     LIFO_Null
33 }LIFO_status;
34
35 //APIs
36 LIFO_status LIFO_Add_item(LIFO_Buf_t *lifo_buf,element_type item);
37 LIFO_status LIFO_get_item(LIFO_Buf_t *lifo_buf,element_type* item);
38 LIFO_status LIFO_init(LIFO_Buf_t *lifo_buf,element_type *buf,unsigned length);
39
```

- LIFO.c File

LIFO_Add_item Function

```

main.c | *lifo.h | lifo.c
2+ * lifo.c
7
8 #include "lifo.h"
9 #include "stdio.h"
10
11 LIFO_status LIFO_Add_item(LIFO_Buf_t *lifo_buf, element_type item)
12 {
13     //Check LIFO is Valid
14     if(!lifo_buf->base || !lifo_buf->head)
15         return LIFO_Null;
16
17
18     //Check LIFO is Full ?
19     // if(lifo_buf->head >= (lifo_buf->base + (lifo_buf->length *4) ))
20     if(lifo_buf->count == lifo_buf->length)
21         return LIFO_full;
22
23     //Add Value
24     *(lifo_buf->head)=item;
25     lifo_buf->head++;
26     lifo_buf->count++;
27
28     return LIFO_no_error;
29
30 }

```

LIFO_get_item Function

```

}
1 LIFO_status LIFO_get_item(LIFO_Buf_t *lifo_buf, element_type* item)
2 {
3     //Check LIFO is Valid
4     if(!lifo_buf->base || !lifo_buf->head)
5         return LIFO_Null;
6
7     //check LIFO is Empty
8     // if(lifo_buf->base == lifo_buf->head);
9     if(lifo_buf->count==0)
10         return LIFO_empty;
11
12     lifo_buf->head --;
13     *item=*(lifo_buf->head);
14     lifo_buf->count--;
15
16     return LIFO_no_error;
17 }

```

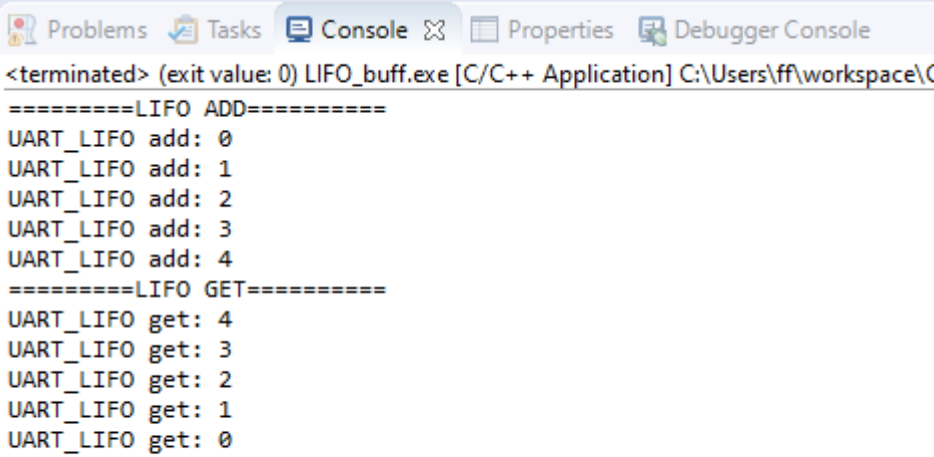
LIFO_init Function

```

1 LIFO_status LIFO_init(LIFO_Buf_t *lifo_buf, element_type *buf, unsigned length)
2 {
3     if(buf==NULL)
4         return LIFO_Null;
5     lifo_buf->base=buf;
6     lifo_buf->head=buf;
7     lifo_buf->length=length;
8     lifo_buf->count=0;
9
10     return LIFO_no_error;
11 }

```

LIFO Output :



The screenshot shows a debugger's console window with tabs for Problems, Tasks, Console, Properties, and Debugger Console. The Console tab is active, displaying the output of a program. The output starts with a terminated status and the file path. It then shows a sequence of 'add' operations to a LIFO stack, followed by 'get' operations that retrieve the elements in reverse order of their addition.

```
<terminated> (exit value: 0) LIFO_buff.exe [C/C++ Application] C:\Users\ff\workspace\C
=====LIFO ADD=====
UART_LIFO add: 0
UART_LIFO add: 1
UART_LIFO add: 2
UART_LIFO add: 3
UART_LIFO add: 4
=====LIFO GET=====
UART_LIFO get: 4
UART_LIFO get: 3
UART_LIFO get: 2
UART_LIFO get: 1
UART_LIFO get: 0
```

Database For Student Using Linked List Data Structure :

- linkedList.h file

```
linked_list.h
2+ | * linked_list.h
7
8 #ifndef LINKED_LIST_H_
9 #define LINKED_LIST_H_
10 #include "stdio.h"
11 #include "stdlib.h"
12 #include "string.h"
13 #include "conio.h"
14
15 #define DPRINTF(...) {fflush(stdout);\
16     fflush(stdin);\
17     printf(__VA_ARGS__);\
18     fflush(stdout);\
19     fflush(stdin);}
20 //effective data
21 struct Sdata
22 {
23     int ID;
24     char name[40];
25     float height ;
26
27 };
28
29 //linked list node
30 struct SStudent
31 {
32     struct Sdata student;
33     struct SStudent* PNextStudent ;
34 };
35
36 struct SStudent* gpFirstStudent;
37
38
39
40
41 void AddStudent();
42 int delete_student ();
43 int view_node();
44 void ReverseList();
45 void lengthOfLinkedList();
46 int Recursion_lengthOfLinkedList(struct SStudent *pSelectedStudent);
47 void view_students();
48 void DeleteAll();
49 void middleList();
50 int viewNodeFromEnd();
51
52 #endif /* LINKED_LIST_H_ */
```


- main.c File

```

1  * main.c
2
3  #include "linked_list.h"
4
5  int main()
6  {
7      char temp_text[40];
8      int count;
9      while(1)
10     {
11         DPRINTF("\n =====");
12         DPRINTF("\n\t Choose on of the Following Option :\n");
13         DPRINTF("\n 1:AddStudent ");
14         DPRINTF("\n 2:DeleteStudent ");
15         DPRINTF("\n 3:viewStudent ");
16         DPRINTF("\n 4: DeleteAll");
17         DPRINTF("\n 5: viewNode ");
18         DPRINTF("\n 6: lengthOfLinkedList ");
19         DPRINTF("\n 7: Recursion_lengthOfLinkedList ");
20         DPRINTF("\n 8: ReverseList ");
21         DPRINTF("\n 9: middleList ");
22         DPRINTF("\n 10: viewNodeFromEnd ");
23         DPRINTF("\n Enter Option Number :");
24         gets(temp_text);
25         DPRINTF("\n =====");
26
27         DPRINTF( "\n ===== ");
28         switch(atoi(temp_text))
29         {
30             case 1:
31                 AddStudent();
32                 break;
33             case 2:
34                 delete_student();
35                 break;
36             case 3:
37                 view_students();
38                 break;
39             case 4:
40                 DeleteAll();
41                 break;
42             case 5:
43                 view_node();
44                 break;
45             case 6:
46                 lengthOfLinkedList();
47                 break;
48             case 7:
49                 count=Recursion_lengthOfLinkedList(gpFirstStudent);
50                 DPRINTF("\nThe Length of Linked List = %d",count);
51                 break;
52             case 8:
53                 ReverseList();
54                 break;
55             case 9:
56                 middleList();
57                 break;
58             case 10:
59                 viewNodeFromEnd();
60                 break;
61         }
62     }
63 }

```

- LinkedList.c File

Add Student Function :

```

12 void AddStudent()
13 {
14     char temp_text[50];
15     struct SStudent* pNewStudent ;
16     struct SStudent* pLastStudent ;
17     //check list is empty ==yes
18     if(gpFirstStudent == NULL)
19     {
20         pNewStudent=(struct SStudent*) malloc (sizeof(struct SStudent));
21         //assign it to gpfirst
22         gpFirstStudent=pNewStudent;
23     }else //list Contains Records
24     {
25         pLastStudent=gpFirstStudent;
26         while (pLastStudent->PNextStudent)
27             pLastStudent=pLastStudent->PNextStudent;
28         pNewStudent=(struct SStudent*) malloc(sizeof(struct SStudent));
29         pLastStudent->PNextStudent=pNewStudent;
30     }
31     //fill new Record
32     DPRINTF("\n Enter The ID :");
33     gets(temp_text);
34     pNewStudent->student.ID=atoi(temp_text);
35
36     DPRINTF("\n Enter Student Full Name :");
37     gets(pNewStudent->student.name);
38
39     DPRINTF("\n Enter Student Height :");
40     gets(temp_text);
41     pNewStudent->student.height=atoi(temp_text);
42
43     //Set The Next Pointer (New_Student) NULL
44     pNewStudent->PNextStudent=NULL;
45

```

Delete Student Function :

```

1 int delete_student ()
2 {
3     char temp_text[40];
4     unsigned int selected_id;
5     //get the The Selected id
6     DPRINTF("\n Enter The Student id to Be Deleted :");
7     gets(temp_text);
8     selected_id=atoi(temp_text);
9
10    //List is not Empty
11    if (gpFirstStudent)
12    {
13        struct SStudent *pSelectedStudent=gpFirstStudent;
14        struct SStudent *pPreviousStudent=NULL;
15
16        //loop on all Records
17        while(pSelectedStudent)
18        {
19            //Compare each Node with The Selected ID
20            if(pSelectedStudent->student.ID ==selected_id)
21            {
22                if(pPreviousStudent)//The first is not The Selected
23                {
24                    pPreviousStudent->PNextStudent=pSelectedStudent->PNextStudent;
25                }else{ //1st Student == ID
26                    gpFirstStudent=pSelectedStudent->PNextStudent;
27                }
28                free(pSelectedStudent);
29                return 1;
30            }
31            pPreviousStudent=pSelectedStudent;
32            pSelectedStudent=pSelectedStudent->PNextStudent;
33        }
34    }
35

```

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 :

```
24 ,
25 void ReverseList()
26 {
27     struct SStudent *pPreviousStudent=NULL;
28     struct SStudent *pCurrentStudent=gpFirstStudent;
29     struct SStudent *pNextStudent=NULL;
30
31     while (pCurrentStudent)
32     {
33         pNextStudent=pCurrentStudent->PNextStudent;
34         pCurrentStudent->PNextStudent=pPreviousStudent;
35         pPreviousStudent=pCurrentStudent;
36         pCurrentStudent=pNextStudent;
37     }
38     gpFirstStudent=pPreviousStudent;
39
40 }
41
42
43
```

Length of LinkedList Function :

```

44
45 void lengthOfLinkedList()
46 {
47     unsigned int count =0;
48     if(gpFirstStudent)
49     {
50         struct SStudent *pSelectedStudent=gpFirstStudent;
51         while(pSelectedStudent)
52         {
53             pSelectedStudent=pSelectedStudent->PNextStudent;
54             count ++;
55         }
56         DPRINTF("\nThe Length of Linked List = %d",count);
57     }
58     else
59     {
60         DPRINTF("\nThe Length of Linked List = %d ",count);
61     }
62 }
63
64
65
66

```

Length of Linked List Using Recursion method Function :

```

67
68 int Recursion_lengthOfLinkedList(struct SStudent *pSelectedStudent)
69 {
70     if(pSelectedStudent)
71     {
72         pSelectedStudent=pSelectedStudent->PNextStudent;
73         return (1+Recursion_lengthOfLinkedList(pSelectedStudent));
74     }
75     else
76         return 0;
77 }
78
79
80
81

```

View Students Function :

```

82
83 void view_students()
84 {
85     struct SStudent* pCurrentStudent =gpFirstStudent;
86     int count =0;
87     if(pCurrentStudent==NULL)
88     {
89         DPRINTF("\n Empty List \n");
90     }
91     else
92     {
93         while(pCurrentStudent)
94         {
95             DPRINTF("\n Record Number %d",count+1);
96             DPRINTF("\n\t ID: %d",pCurrentStudent->student.ID);
97             DPRINTF("\n\t Name: %s",pCurrentStudent->student.name);
98             DPRINTF("\n\t Height: %f",pCurrentStudent->student.height);
99             pCurrentStudent=pCurrentStudent->PNextStudent;
100             count++;
101         }
102     }
103 }
104
105
106
107

```

Delete All Function :

```
1
2 void DeleteAll()
3 {
4     struct SStudent* pCurrentStudent =gpFirstStudent;
5     if(pCurrentStudent==NULL)
6     {
7         DPRINTF("\n Empty List \n");
8     }
9     else
10    {
11        while(pCurrentStudent)
12        {
13            struct SStudent* pTempStudent =pCurrentStudent;
14            pCurrentStudent=pCurrentStudent->PNextStudent;
15            free(pTempStudent);
16        }
17        gpFirstStudent=NULL;
18    }
19 }
```

The Index of The Middle of List Function :

```
1
2 void middleList()
3 {
4     unsigned int length ,middle;
5     struct SStudent* pCurrentStudent =gpFirstStudent;
6     if(pCurrentStudent==NULL)
7     {
8         DPRINTF("\n Empty List \n");
9     }
10    length =Recursion_lengthOfLinkedList(pCurrentStudent);
11    middle =(length /2);
12
13    DPRINTF("\n The middle =%d",middle);
14 }
```

View Node From End Function :

```
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
```

view Students
After Reversing

```
=====
```

```
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
```

```
-----
```

linked_list_student_projects.c | C++ Application | C++3 in Windows | C++ Programming | linked_

```
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 one 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
```

View Student After Use Function
Delete_All

```
=====
Empty List ←
```