Unit 4 Lesson 1

Guirguis Hedia

FIFO DATA Structure :

- main.c File

```c
* main.c

#include "fifo.h"


void main ()
{
    FIFO_Buf_t FIFO_UART;
    element_type i,temp;
    if(FIFO_init(&FIFO_UART,uart_buffer,5)==FIFO_no_error)
        printf("FIFO init ---Done \n");

    for (i=0;i<7;i++)
    {
        printf("FIFO Enqueue (%x) \n",i);
        if(FIFO_enqueue(&FIFO_UART,i)==FIFO_no_error)
            printf("FIFO enqueue -----Done \n");
        else
            printf("FIFO enqueue ----- Failed\n");
    }

    FIFO_print(&FIFO_UART);


    if(FIFO_dequeue(&FIFO_UART,&temp)==FIFO_no_error)
        printf("FIFO dequeue %x -----Done \n",temp);

    if(FIFO_dequeue(&FIFO_UART,&temp)==FIFO_no_error)
        printf("FIFO dequeue %x -----Done \n",temp);

    FIFO_print(&FIFO_UART);

}
```

- FIFO.h File

fifo.c | fifo.h | main.c

```c
* fifo.h

#ifndef FIFO_H_
#define FIFO_H_

#include "stdio.h"
#include "stdint.h"

//USER Configuration
//Select The Element Type (uint8_t , uint16_t , uint32_t , ....)
#define element_type uint8_t

//Create buffer
#define width 5
element_type uart_buffer [width];

//FIFO Data Types
typedef struct {
    unsigned int length;
    unsigned int count ;
    element_type* head ;
    element_type* tail ;
    element_type* base ;
}FIFO_Buf_t;

typedef enum{
    FIFO_no_error,
    FIFO_full,
    FIFO_empty,
    FIFO_null
}FIFO_Buf_Status;


//FIFO APIs
FIFO_Buf_Status FIFO_init(FIFO_Buf_t *fifo,element_type* buf,uint32_t length);
FIFO_Buf_Status FIFO_enqueue (FIFO_Buf_t *fifo ,element_type item);
FIFO_Buf_Status FIFO_dequeue (FIFO_Buf_t *fifo ,element_type* item);
FIFO_Buf_Status FIFO_IS_FULL (FIFO_Buf_t *fifo );
void FIFO_print (FIFO_Buf_t *fifo );

#endif /* FIFO_H_ */
```

Problems | Tasks | Console | Properties | Debugger Console

- FIFO.c File

FIFO_init() Function And FIFO_enqueue() Function :

```c


FIFO_Buf_Status FIFO_init(FIFO_Buf_t *fifo,element_type* buf,uint32_t length)
{
    if(buf==NULL)
        return FIFO_null;

    fifo->base =buf;
    fifo->head =buf;
    fifo->tail =buf;
    fifo->length =length;
    fifo->count =0;

    return FIFO_no_error;

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

    if(FIFO_IS_FULL(fifo)==FIFO_full)
        return FIFO_full;

    *(fifo->head)=item;
    fifo->count++;

    //circular FIFO
    if(fifo->head ==(fifo->base +(fifo->length *sizeof(element_type))))
        fifo->head=fifo->base;
    fifo->head++;

    return FIFO_no_error;
}
```

FIFO_dequeue() Function And FIFO_is_FULL() Function :

```c
}
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;

}
void FIFO_print (FIFO_Buf_t *fifo )
```

FIFO_print() Function :

```c
}
void FIFO_print (FIFO_Buf_t *fifo )
{
    int i;
    element_type* temp;

    if(fifo->count ==0)
    {
        printf("fifo is empty \n");
    }else{
        temp =fifo->tail;
        printf("\n=======FIFO_print=====\n");
        for(i=0;i<fifo->count;i++)
        {
            printf("\t %x\n",*temp);
            temp++;
        }
        printf("\n============\n");
    }
}
```

FIFO Output :

Problems | Tasks | Console | Properties | Debugger Console

```
<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

```c

#include "lifo.h"

int  main()
{
    element_type i,temp=0 ;

    LIFO_Buf_t uart_lifo ,I2C_lifo;
    //static allocation
    LIFO_init(&uart_lifo,buffer,5);

    //dynamic allocation
    element_type* buffer2=(element_type*)malloc(5 *sizeof(unsigned int));
    LIFO_init(&I2C_lifo,buffer2,5);
    printf("=========LIFO ADD==========\n");
    for (i=0 ;i<8;i++)
    {
        if(LIFO_Add_item(&uart_lifo,i)==LIFO_no_error)
        printf("UART_LIFO add: %d \n",i);
    }
    printf("=========LIFO GET==========\n");
    for (i=0 ;i<8;i++)
    {
        if(LIFO_get_item(&uart_lifo,&temp)==LIFO_no_error)
        printf("UART_LIFO get: %d \n",temp);
    }

    return 0;

}

```

- LIFO.h file

```c
 * lifo.h

#ifndef LIFO_H_
#define LIFO_H_
#include "stdlib.h"
#include "stdint.h"

//type Definitions
#define element_type    uint8_t
#define width 5
element_type buffer[width];


//Select The Element Type (uint8_t , uint16_t , uint32_t,.....)

typedef struct {
    unsigned int length ;
    unsigned int count ;
    element_type *base ;
    element_type* head ;
}LIFO_Buf_t;

typedef enum{
    LIFO_no_error,
    LIFO_full,
    LIFO_empty,
    LIFO_Null
}LIFO_status;

//APIs
LIFO_status LIFO_Add_item(LIFO_Buf_t *lifo_buf,element_type item);
LIFO_status LIFO_get_item(LIFO_Buf_t *lifo_buf,element_type* item);
LIFO_status LIFO_init(LIFO_Buf_t *lifo_buf,element_type *buf,unsigned length);
```

- LIFO.c File

LIFO_Add_item Function

main.c | *lifo.h | lifo.c

```c
 * lifo.c

#include "lifo.h"
#include "stdio.h"

LIFO_status LIFO_Add_item(LIFO_Buf_t *lifo_buf,element_type item)
{
    //Check LIFO is Valid
    if(!lifo_buf->base || !lifo_buf->head)
        return LIFO_Null;


    //Check LIFO is FUll ?
//  if(lifo_buf->head >= (lifo_buf->base + (lifo_buf->length *4) ))
    if(lifo_buf->count ==lifo_buf->length)
        return LIFO_full;

    //Add Value
    *(lifo_buf->head)=item;
    lifo_buf->head++;
    lifo_buf->count++;

    return LIFO_no_error;

}
```

LIFO_get_item Function

```c
}
LIFO_status LIFO_get_item(LIFO_Buf_t *lifo_buf,element_type* item)
{
    //Check LIFO is Valid
    if(!lifo_buf->base || !lifo_buf->head)
        return LIFO_Null;

    //check LIFO is Empty
//  if(lifo_buf->base == lifo_buf->head);
    if(lifo_buf->count==0)
        return LIFO_empty;

    lifo_buf->head --;
    *item=*(lifo_buf->head);
    lifo_buf->count--;

    return LIFO_no_error;
}
```

LIFO_init Function

```c
LIFO_status LIFO_init(LIFO_Buf_t *lifo_buf,element_type *buf,unsigned length)
{
    if(buf==NULL)
        return LIFO_Null;
    lifo_buf->base=buf;
    lifo_buf->head=buf;
    lifo_buf->length=length;
    lifo_buf->count=0;

    return LIFO_no_error;
}
```

LIFO Output :

Problems Tasks Console Properties Debugger Console

```
<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

```c
* 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 :

```c
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 :

```c
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 :

```c
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 :

```c
}

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 :

```c
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 :

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

}
```

View Students Function :

```c
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 :

```c
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 :

```c
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