**Batch: G2 Roll No.: 16010421028**

**Experiment / assignment / tutorial No. 9**

**Grade: AA / AB / BB / BC / CC / CD /DD**

**Signature of the Staff In-charge with date**

|  |
| --- |
| **TITLE:**  Dynamic Memory Allocation. |

**AIM:** Program to demonstrate dynamic memory allocation using malloc() & free () function.

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Expected OUTCOME of Experiment:**

CO4: Design modular programs using functions and demonstrate the concept of pointers and file handling.

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Books/ Journals/ Websites referred:**

1. Programming in C, second edition, Pradeep Dey and Manas Ghosh, Oxford University Press.
2. Programming in ANSI C, fifth edition, E Balagurusamy, Tata McGraw Hill.
3. Introduction to programming and problem solving , G. Michael Schneider ,Wiley India edition.
4. [**http://cse.iitkgp.ac.in/~rkumar/pds-vlab/**](http://cse.iitkgp.ac.in/~rkumar/pds-vlab/)

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Problem Definition:**

Implementing a C program to create a student list of a class using Dynamic memory allocation. It will have the details of students as roll number and name. Program should support the following operations (menu driven).

1. Insert

2. Delete

3. Display

use malloc for insert and free for delete

**Algorithm:**

* Import standard library with the command (#include <stdlib.h>)
* Declare 2 Structures
* Assign A Pointer to A Structure
* Declare a char pointer
* Now allocating memory to each pointer using dynamic memory allocation (malloc () function)
* Input the value to be stored in the 2 data types of the structure
* Use switch case to show menu driven output
* Print 3 Choices and Input from The User
* For Display Function

1. Display The Data Types in An Orderly Manner.

* For Delete Function

a. Delete The Entire Elements of The Structure Using Free Function.

* For Insert Function

1. Take the input for each of the players. Store it in the structured array.

**Implementation details:**

#include<stdio.h>

#include<string.h>

#include<stdlib.h>

int main()

{

int i,n,roll,ch;

char name[20];

struct student

{

int roll;

char name[20];

}\*ptr,\*temp;

printf("Enter the number of students:");

scanf("%d",&n);

ptr=(struct student\*)malloc(n\*sizeof(struct student));

for(i=0;i<n;i++)

{

printf("Enter the roll number:");

scanf("%d",&ptr[i].roll);

printf("Enter the name:");

scanf("%s",ptr[i].name);

}

do{

printf("\n1.Insert\n2.Delete\n3.Display\n4.Exit");

printf("\n Enter your choice:");

scanf("%d",&ch);

switch(ch)

{

case 1:

printf("Enter the roll number:");

scanf("%d",&roll);

printf("Enter the name:");

scanf("%s",name);

temp=(struct student\*)malloc(n\*sizeof(struct student));

for(i=0;i<n;i++)

{

temp[i].roll=ptr[i].roll;

strcpy(temp[i].name,ptr[i].name);

}

temp[n].roll=roll;

strcpy(temp[n].name,name);

free(ptr);

ptr=temp;

n++;

for(i=0;i<n;i++)

{

printf("%d %s\n",ptr[i].roll,ptr[i].name);

}

break;

case 2:

printf("Enter the roll number:");

scanf("%d",&roll);

temp=(struct student\*)malloc(n\*sizeof(struct student));

for(i=0;i<n;i++)

{

if(ptr[i].roll==roll)

{

i++;

}

temp[i-1].roll=ptr[i].roll;

strcpy(temp[i-1].name,ptr[i].name);

}

free(ptr);

ptr=temp;

n--;

for(i=0;i<n;i++)

{

printf("%d %s\n",ptr[i].roll,ptr[i].name);

}

break;

case 3:

for(i=0;i<n;i++)

{

printf("%d %s\n",ptr[i].roll,ptr[i].name);

}

break;

}

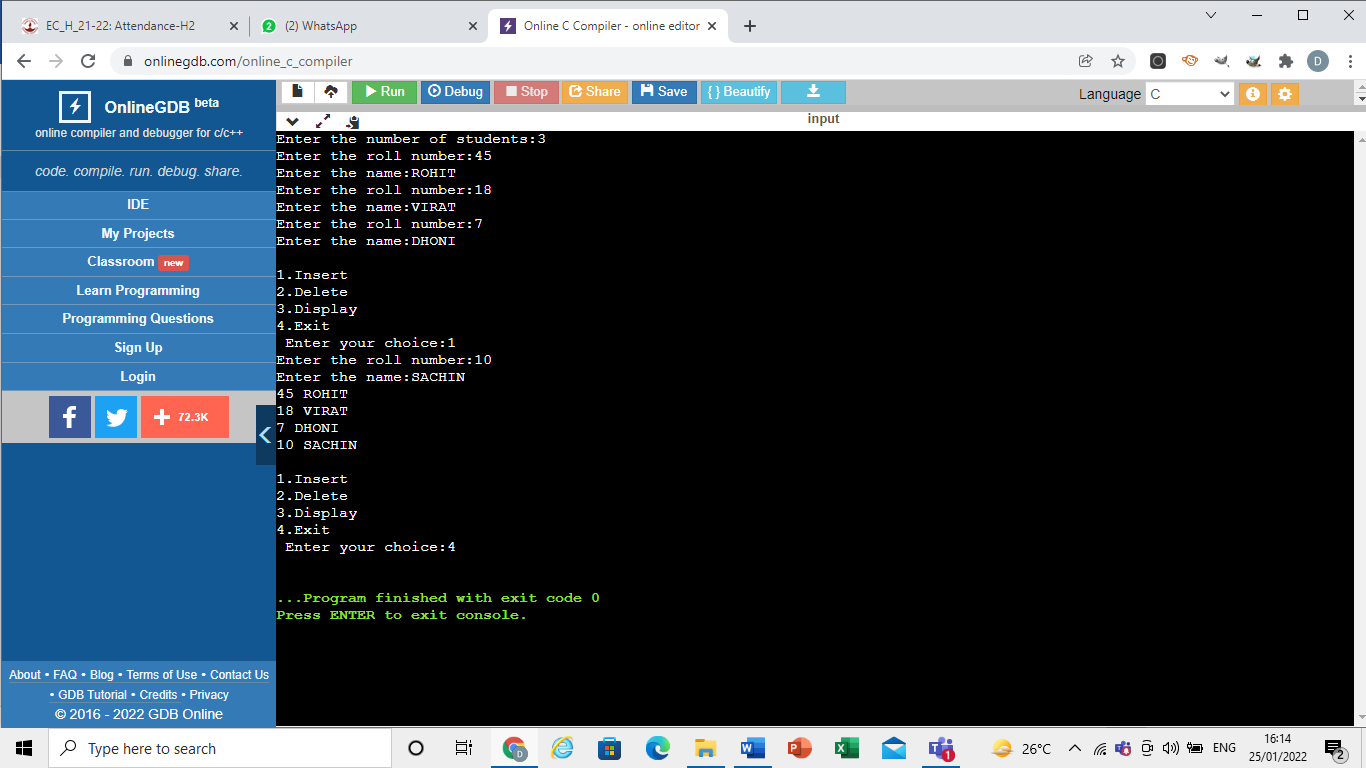
}while(ch!=4);

return 0;

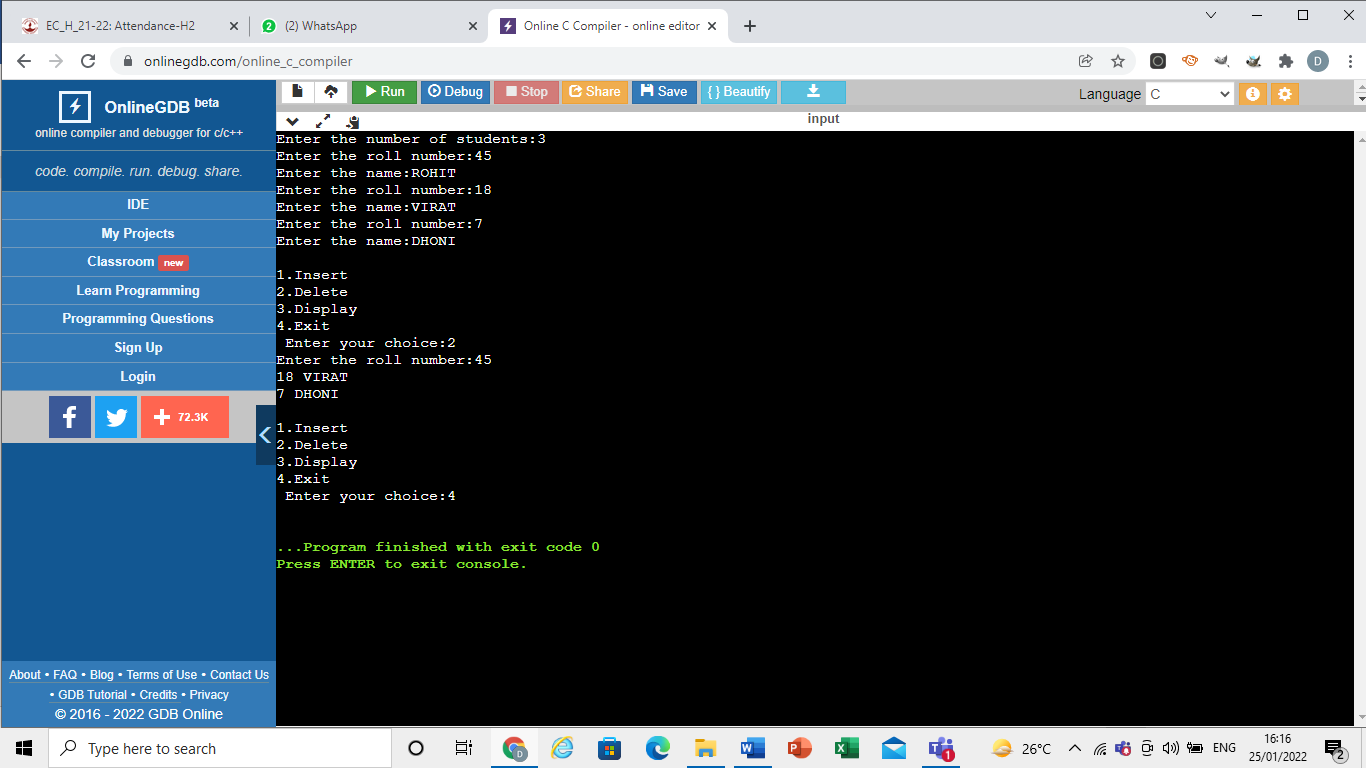
}

**Output(s):**

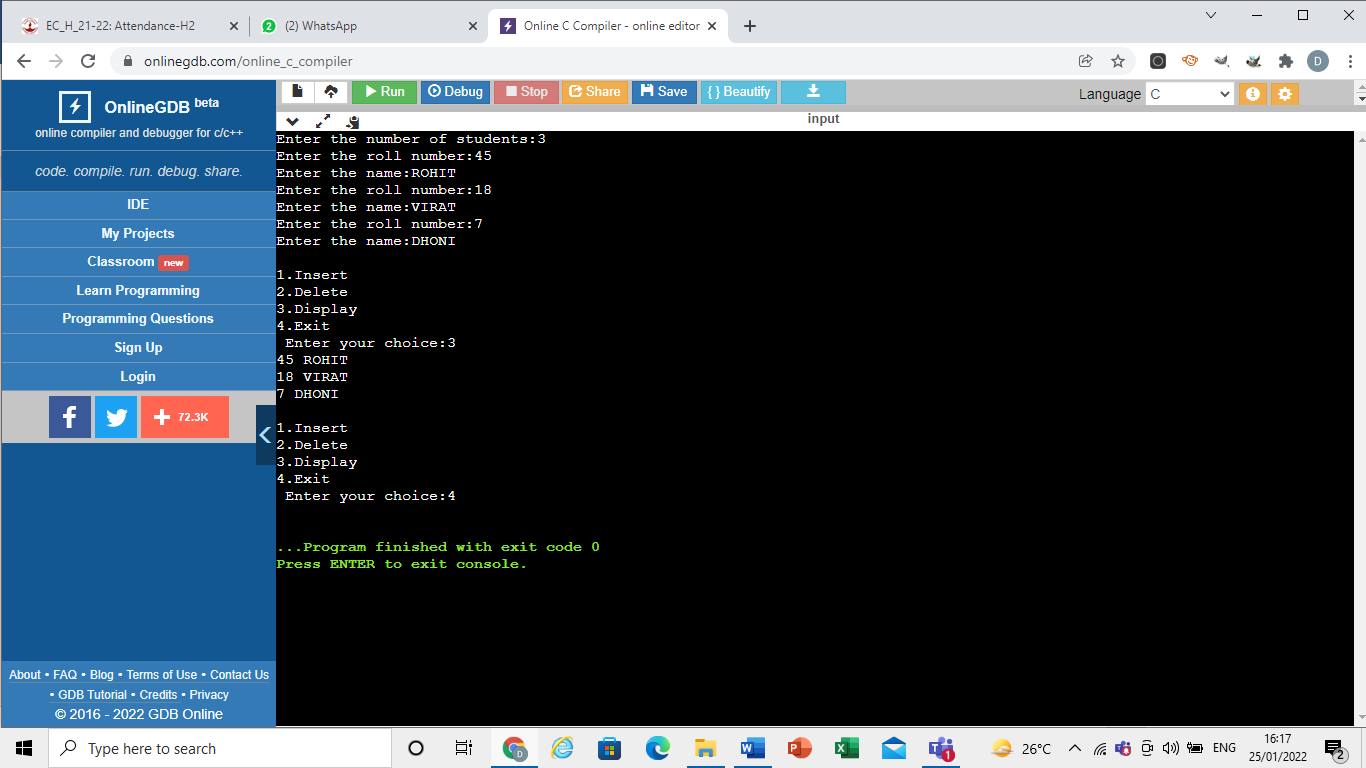
1 INSERT:



2 DELETE:



3 DISPLAY:



**Conclusion:**

In the above code:

* We learned what a pointer is and how to use it
* We learned how to use dynamic memory allocation and how to use malloc () function
* We learned what is the free () function and how to use it.

**Post Lab Descriptive Questions**

1. **What is the difference between malloc and Calloc?**

|  |  |
| --- | --- |
| Malloc () | Calloc () |
| Malloc () allocates a single block of the memory required in bytes. | Calloc () allocates multiple blocks of the requested memory to any process or program dynamically. |
| void \*malloc (size\_t size); | void \*calloc (size\_t num, size\_t size); |
| the malloc () function fails to initialize and clear out the allocated memory. It contains a random garbage value and the allocated memory item is incapable of being altered. | The Calloc () function initialises the earlier allocated memory to zero. |
| The malloc () performs quickly and without any lags. | The calloc () is comparatively slow and takes a longer time to complete as it first initialises the value of the pointer to ZERO. However, this lag is minuscule and does not affect the performance of the program in any manner. |
| The malloc () function assigns memory of the desired 'size' from the available heap. | The Calloc () function assigns memory that is the size of what’s equal to ‘num \*size’. |
| The name malloc is attributed to memory allocation. | The name calloc means contiguous allocation. |
| void \*malloc (size\_t n) will return a pointer to get a value of n bytes of uninitialized storage if the request is satisfied. Conversely, it will give it a NULL value if the request fails to be successful. In case the memory space allocated by malloc () gets overrun, then the results will remain as undefined. | Void \*Calloc (size\_t n, size\_t size) will return a pointer to allocate enough free space of an array of n objects; which will be of a pre-defined and specified size. The NULL value is assigned in case the request fails to be satisfied. Here, the storage is again initialised to zero |
| Malloc () takes in just one argument – the number of bytes. | Calloc () takes in two arguments – the number of blocks as well as the size of every single block. |

1. **Consider the following C code. What will be the output?**

# include<stdio.h>

# include<stdlib.h>

void fun (int \*a)

{

  a = (int\*) malloc(sizeof(int));

}

int main ()

{

  int \*p;

  fun(p);

  \*p = 6;

  printf("%d\n",\*p);

  return (0);

}

(A) Compiler Error

(B) 6

(C) Runtime Error

(D) Garbage Value

Ans: (D) Garbage Value

**3) Difference between Static and Dynamic Memory allocation**

|  |  |
| --- | --- |
| Static Memory Allocation | Dynamic Memory Allocation |
| In the static memory allocation, variables get allocated permanently, till the program executes or function call finishes. | In the Dynamic memory allocation, variables get allocated only if your program unit gets active. |
| Static Memory Allocation is done before program execution. | Dynamic Memory Allocation is done during program execution |
| It uses [stack](https://www.geeksforgeeks.org/stack-data-structure/) for managing the static allocation of memory | It uses [heap](https://www.geeksforgeeks.org/heap-data-structure/) for managing the dynamic allocation of memory |
| It is less efficient | It is more efficient |
| In Static Memory Allocation, there is no memory re-usability | In Dynamic Memory Allocation, there is memory re-usability and memory can be freed when not required |
| In static memory allocation, once the memory is allocated, the memory size can not change. | In dynamic memory allocation, when memory is allocated the memory size can be changed. |
| In this memory allocation scheme, we cannot reuse the unused memory. | This allows reusing the memory. The user can allocate more memory when required. Also, the user can release the memory when the user needs it. |
| In this memory allocation scheme, execution is faster than dynamic memory allocation. | In this memory allocation scheme, execution is slower than static memory allocation |
| In this memory is allocated at compile time. | In this memory is allocated at run time. |
| In this allocated memory remains from start to end of the program. | In this allocated memory can be released at any time during the program. |
| **Example:** This static memory allocation is generally used for [array](https://www.geeksforgeeks.org/introduction-to-arrays/). | **Example:** This dynamic memory allocation is generally used for [linked list](https://www.geeksforgeeks.org/data-structures/linked-list/). |

**Date: \_\_\_\_\_\_\_\_\_\_ Signature of faculty in-charge**