Course Name:	**Programming in C**	**Semester:**	**II**
Date of Performance:	**16/04/2025**	**DIV/ Batch No:**	**c2-2**
Student Name:	**Ashwera Hasan**	**Roll No:**	**16010124107**

Experiment No: 8
Title: Pointers and dynamic memory allocation

Aim and Objective of the Experiment:

Write a program in C to demonstrate pointers and dynamic memory allocation

COs to be achieved:

CO5: Apply concepts of pointers in dynamic memory allocation and file handling.

Theory:

1. Pointers: A pointer is a variable that stores the memory address of another variable. It provides a way to indirectly access and manipulate the data stored in memory.

- Syntax: int *ptr; // Declares a pointer to an integer
- Usage:
  - Address-of Operator (`&`): Used to get the memory address of a variable.
```
int x = 10;
int *ptr = &x;  // ptr now holds the address of x
```
  - Dereference Operator (`*`): Used to access the value stored at the memory address held by the pointer.
```
int y = *ptr;  // y now holds the value 10
```
- Pointer Arithmetic:
  Pointers can be incremented or decremented to navigate through arrays or memory blocks.
```
int arr[3] = {10, 20, 30};
int *ptr = arr;  // Points to the first element of the array
ptr++;  // Now points to the second element
```

2. Dynamic Memory Allocation:
Dynamic memory allocation allows programs to request memory from the heap at runtime, rather than at compile time. This is useful when the amount of memory needed is not known in advance.
malloc: Allocates a block of memory of a specified size and returns a pointer to the beginning of the block.
```
int *ptr = (int *)malloc(10 * sizeof(int));  // Allocates memory for 10 integers
```
calloc: Allocates memory for an array of elements, initializes them to zero, and returns a pointer to the memory.
```
int *ptr = (int *)calloc(10, sizeof(int));  // Allocates and initializes memory for 10 integers
```

realloc: Resizes a previously allocated block of memory.
ptr = (int *)realloc(ptr, 20 * sizeof(int)); // Resizes the allocated memory to hold 20 integers
free: Deallocates a block of memory previously allocated by `malloc`, `calloc`, or `realloc`.
free(ptr); // Deallocates the memory block

- Memory Management:
- Heap vs. Stack:
  - Stack: Memory is automatically managed (allocated and deallocated) for local variables.
  - Heap: Memory must be manually managed by the programmer using dynamic memory allocation functions.
  - Memory Leaks: Occur when dynamically allocated memory is not properly deallocated, leading to wasted memory resources.
  - Dangling Pointers: Occur when a pointer points to a memory location that has been deallocated, leading to undefined behavior.

3. Practical Applications:
- Dynamic Arrays: Arrays whose size can be determined at runtime.

```
int n;
scanf("%d", &n);
int *arr = (int *)malloc(n * sizeof(int));
```

- Linked Lists, Trees, and Graphs: Data structures that require dynamic memory allocation for nodes.
- String Manipulation: Dynamically allocated strings that can grow or shrink as needed.

4. Best Practices:
- Always check if memory allocation was successful (i.e., the pointer is not `NULL`).
- Always free dynamically allocated memory when it is no longer needed to avoid memory leaks.
- Avoid dereferencing null or dangling pointers to prevent runtime errors.

Using pointers and dynamic memory allocation, you can write more flexible and efficient C programs that can handle varying amounts of data and complex data structures.

Problem Statements:	Conclusion:
Dynamic Memory Management for a Student Database System **Background:** In a university, there is a need to manage student records efficiently. Each student record consists of the following details: • Student ID (integer) • Name (string) • Age (integer) • GPA (float) The number of students is not fixed and can vary over time. Therefore, the system should be able to dynamically allocate and deallocate memory for student records as needed.	

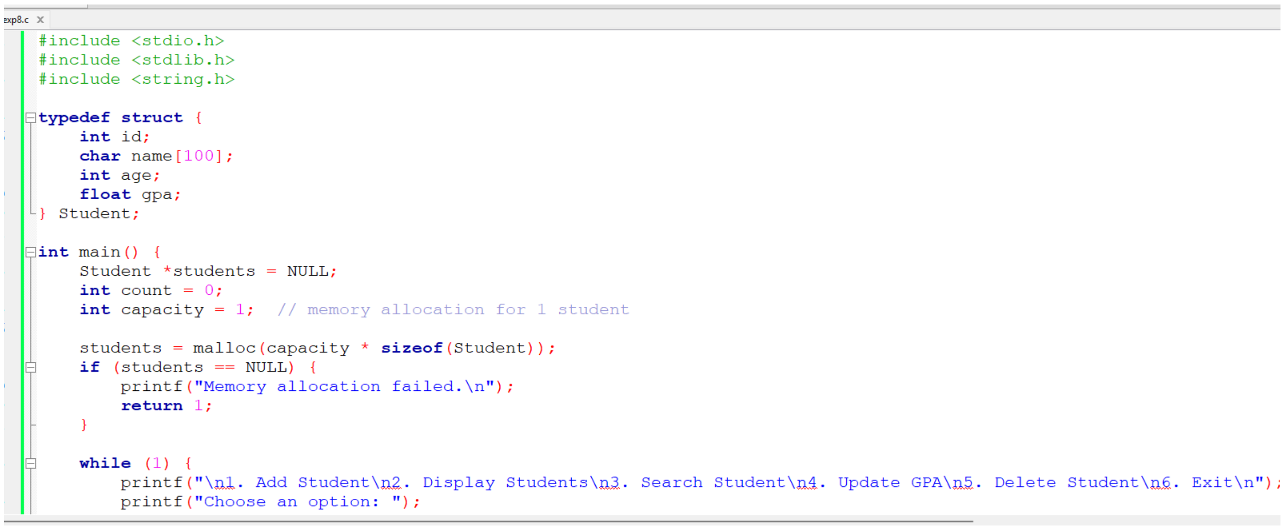
Problem:
Design and implement a C program that manages a dynamic student database using pointers and dynamic memory allocation. The program should provide the following functionalities:

1. **Add a Student:**
    - Prompt the user to enter the details of a new student (Student ID, Name, Age, GPA).
    - Dynamically allocate memory to store the student record.
    - Add the student record to the database.
2. **Display All Students:**
    - Display the details of all students currently stored in the database.
3. **Search for a Student by ID:**
    - Prompt the user to enter a Student ID.
    - Search the database for the student with the given ID.
    - Display the student's details if found; otherwise, display a "Student not found" message.
4. **Update a Student's GPA:**
    - Prompt the user to enter a Student ID.
    - Search the database for the student with the given ID.
    - If found, prompt the user to enter the new GPA and update the student's record.
    - If not found, display a "Student not found" message.
5. **Delete a Student:**
    - Prompt the user to enter a Student ID.
    - Search the database for the student with the given ID.
    - If found, delete the student's record and deallocate the memory.
    - If not found, display a "Student not found" message.
6. **Exit:**
    - Deallocate all dynamically allocated memory before exiting the program.
    - Display a message indicating that the program has exited successfully.

Requirements:

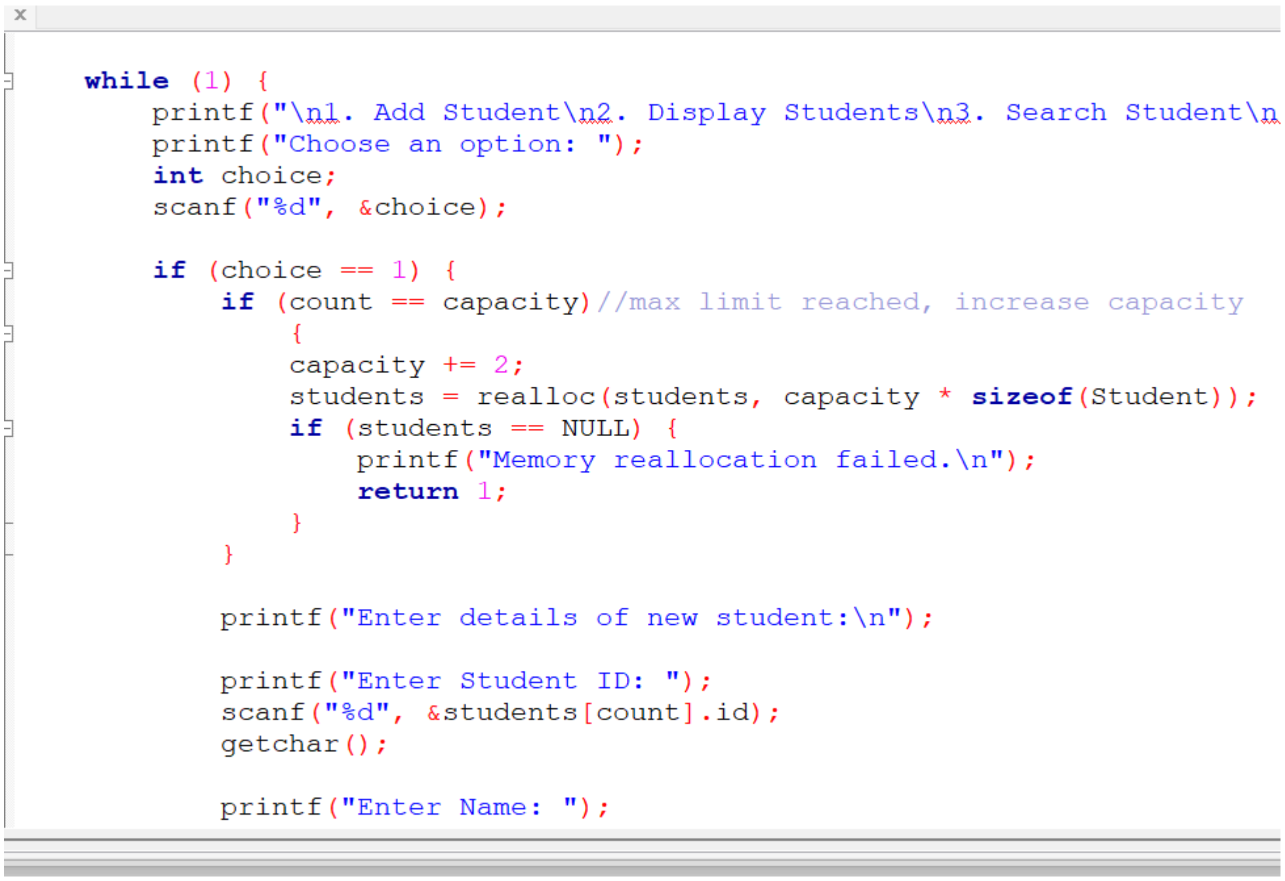
- Use **pointers** to manage the student records.
- Use **dynamic memory allocation** (e.g., malloc, calloc, realloc, free) to allocate and deallocate memory for student records.
- Ensure that the program handles memory allocation failures gracefully (e.g., by displaying an error message and exiting if memory allocation fails).
- Implement a menu-driven interface that allows the user to choose between the different functionalities.
- The program should continue running until the user chooses to exit.

Code :

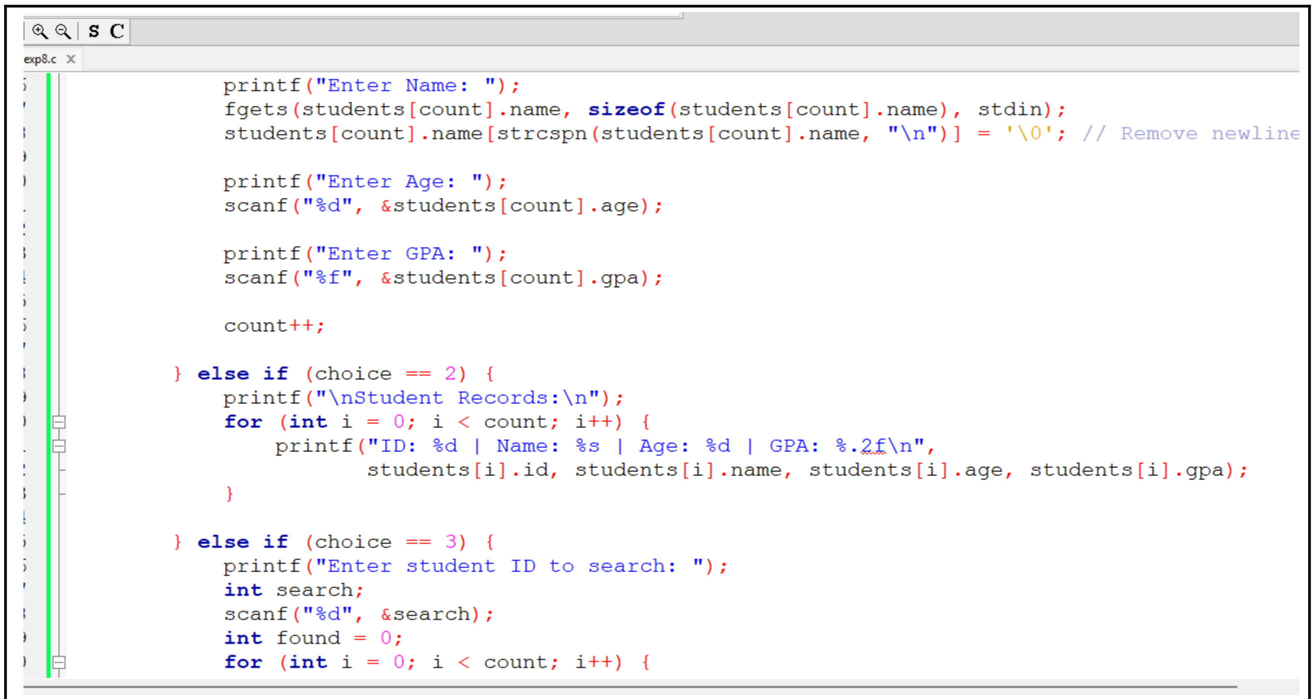


exp8.c

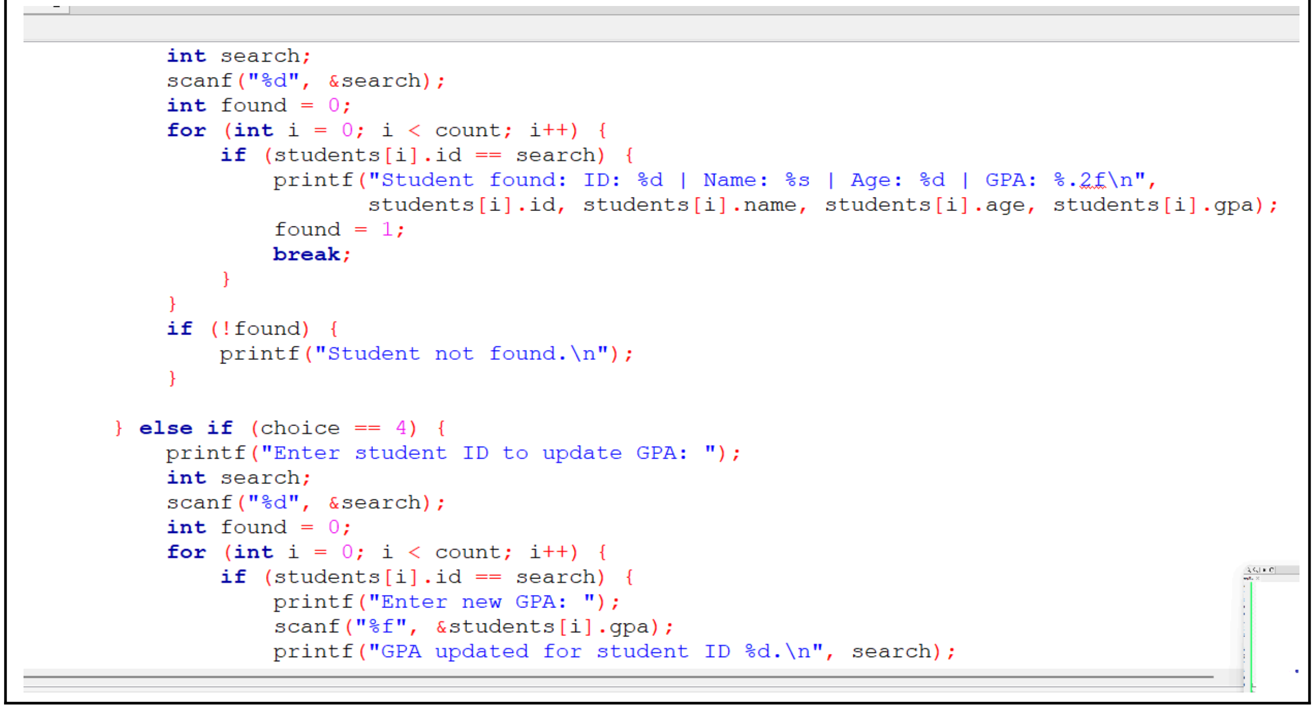
```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

typedef struct {
    int id;
    char name[100];
    int age;
    float gpa;
} Student;

int main() {
    Student *students = NULL;
    int count = 0;
    int capacity = 1;  // memory allocation for 1 student

    students = malloc(capacity * sizeof(Student));
    if (students == NULL) {
        printf("Memory allocation failed.\n");
        return 1;
    }

    while (1) {
        printf("\n1. Add Student\n2. Display Students\n3. Search Student\n4. Update GPA\n5. Delete Student\n6. Exit\n");
        printf("Choose an option: ");
```



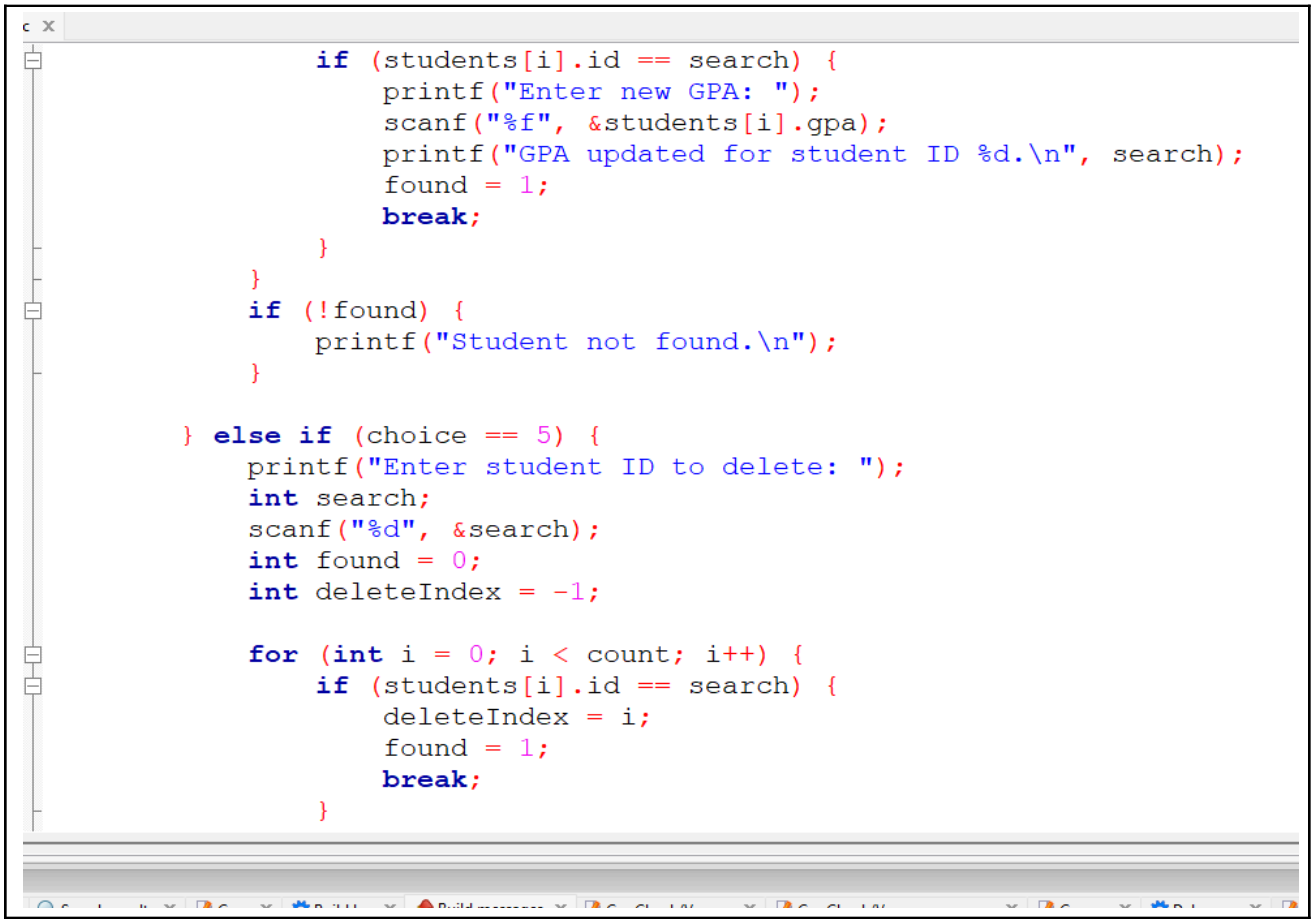
```
    while (1) {
        printf("\n1. Add Student\n2. Display Students\n3. Search Student\n
        printf("Choose an option: ");
        int choice;
        scanf("%d", &choice);

        if (choice == 1) {
            if (count == capacity)//max limit reached, increase capacity
                {
                capacity += 2;
                students = realloc(students, capacity * sizeof(Student));
                if (students == NULL) {
                    printf("Memory reallocation failed.\n");
                    return 1;
                }
            }

            printf("Enter details of new student:\n");

            printf("Enter Student ID: ");
            scanf("%d", &students[count].id);
            getchar();

            printf("Enter Name: ");
```



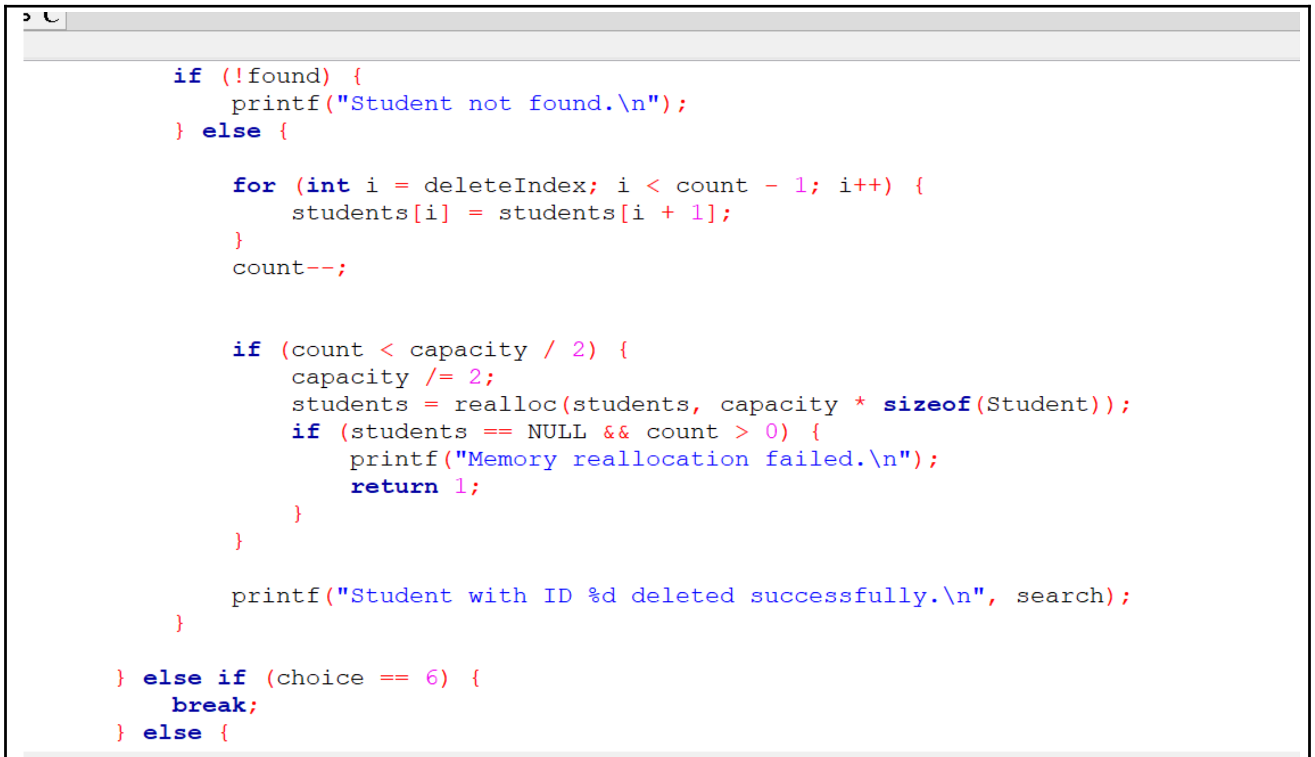
```
            printf("Enter Name: ");
            fgets(students[count].name, sizeof(students[count].name), stdin);
            students[count].name[strcspn(students[count].name, "\n")] = '\0'; // Remove newline

            printf("Enter Age: ");
            scanf("%d", &students[count].age);

            printf("Enter GPA: ");
            scanf("%f", &students[count].gpa);

            count++;

        } else if (choice == 2) {
            printf("\nStudent Records:\n");
            for (int i = 0; i < count; i++) {
                printf("ID: %d | Name: %s | Age: %d | GPA: %.2f\n",
                       students[i].id, students[i].name, students[i].age, students[i].gpa);
            }

        } else if (choice == 3) {
            printf("Enter student ID to search: ");
            int search;
            scanf("%d", &search);
            int found = 0;
            for (int i = 0; i < count; i++) {
```



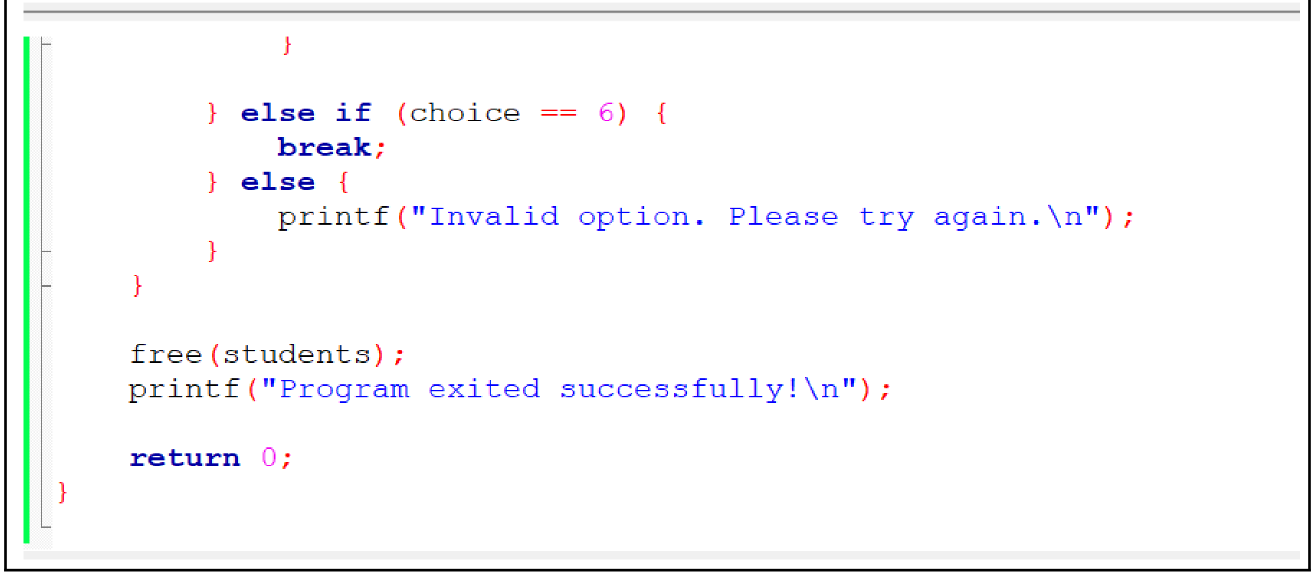
```
            int search;
            scanf("%d", &search);
            int found = 0;
            for (int i = 0; i < count; i++) {
                if (students[i].id == search) {
                    printf("Student found: ID: %d | Name: %s | Age: %d | GPA: %.2f\n",
                           students[i].id, students[i].name, students[i].age, students[i].gpa);
                    found = 1;
                    break;
                }
            }
            if (!found) {
                printf("Student not found.\n");
            }

        } else if (choice == 4) {
            printf("Enter student ID to update GPA: ");
            int search;
            scanf("%d", &search);
            int found = 0;
            for (int i = 0; i < count; i++) {
                if (students[i].id == search) {
                    printf("Enter new GPA: ");
                    scanf("%f", &students[i].gpa);
                    printf("GPA updated for student ID %d.\n", search);
```



```
                if (students[i].id == search) {
                    printf("Enter new GPA: ");
                    scanf("%f", &students[i].gpa);
                    printf("GPA updated for student ID %d.\n", search);
                    found = 1;
                    break;
                }
            }
            if (!found) {
                printf("Student not found.\n");
            }

        } else if (choice == 5) {
            printf("Enter student ID to delete: ");
            int search;
            scanf("%d", &search);
            int found = 0;
            int deleteIndex = -1;

            for (int i = 0; i < count; i++) {
                if (students[i].id == search) {
                    deleteIndex = i;
                    found = 1;
                    break;
                }
```

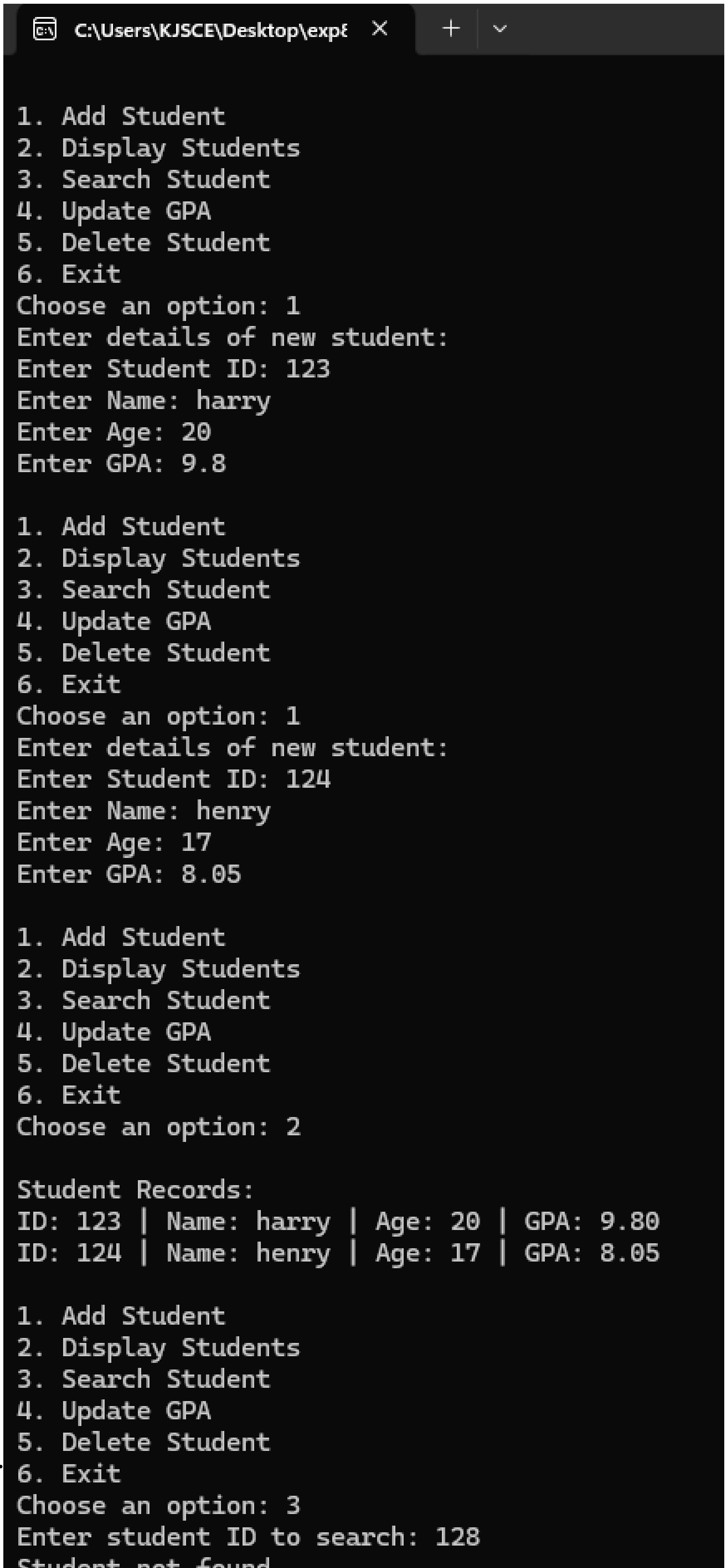


```
        if (!found) {
            printf("Student not found.\n");
        } else {

            for (int i = deleteIndex; i < count - 1; i++) {
                students[i] = students[i + 1];
            }
            count--;


            if (count < capacity / 2) {
                capacity /= 2;
                students = realloc(students, capacity * sizeof(Student));
                if (students == NULL && count > 0) {
                    printf("Memory reallocation failed.\n");
                    return 1;
                }
            }

            printf("Student with ID %d deleted successfully.\n", search);
        }

    } else if (choice == 6) {
        break;
    } else {
```

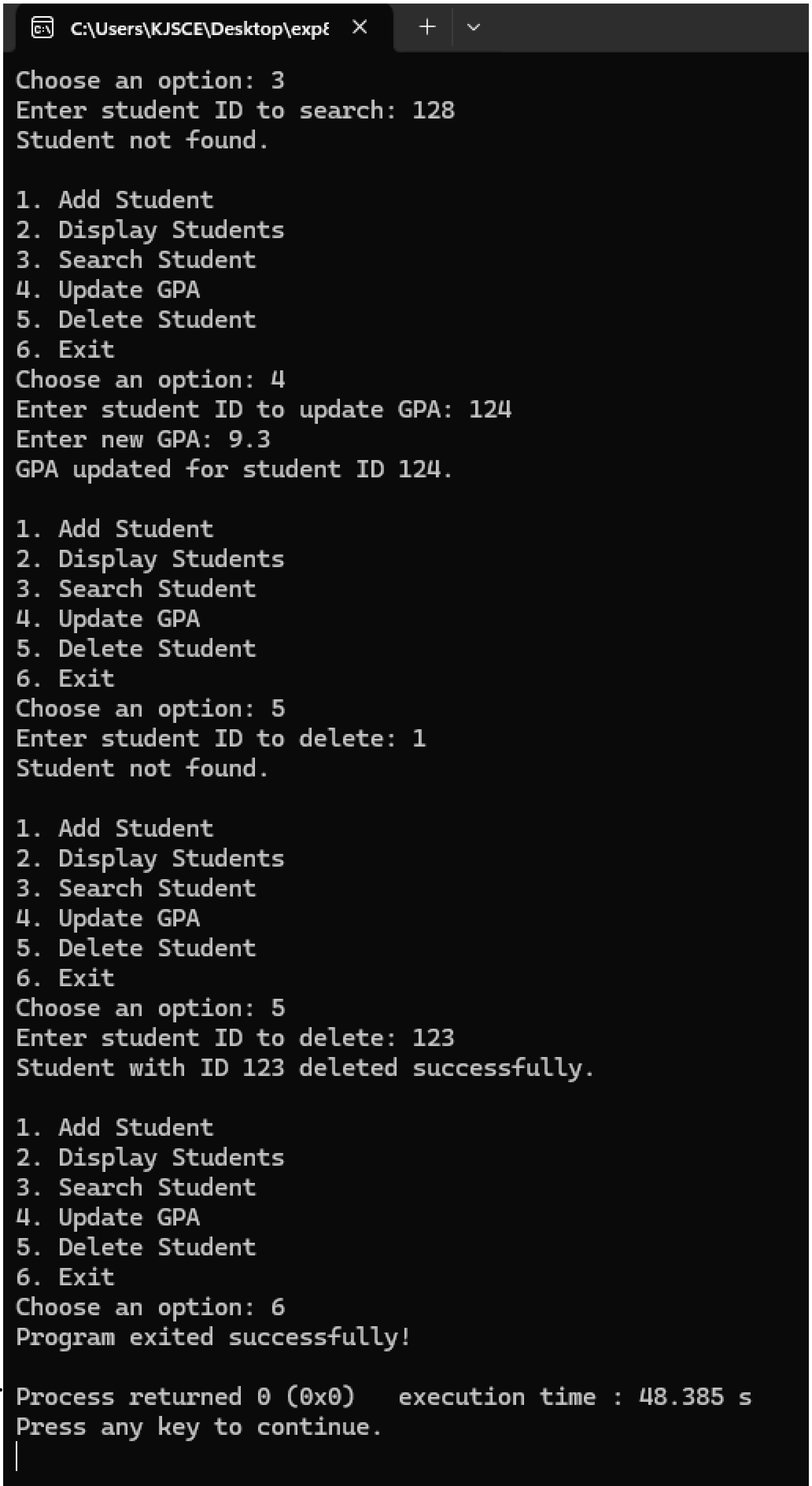


```
                }

            } else if (choice == 6) {
                break;
            } else {
                printf("Invalid option. Please try again.\n");
            }
        }

        free(students);
        printf("Program exited successfully!\n");

        return 0;
}
```

Output:



```
C:\Users\KJSCE\Desktop\exp8

1. Add Student
2. Display Students
3. Search Student
4. Update GPA
5. Delete Student
6. Exit
Choose an option: 1
Enter details of new student:
Enter Student ID: 123
Enter Name: harry
Enter Age: 20
Enter GPA: 9.8

1. Add Student
2. Display Students
3. Search Student
4. Update GPA
5. Delete Student
6. Exit
Choose an option: 1
Enter details of new student:
Enter Student ID: 124
Enter Name: henry
Enter Age: 17
Enter GPA: 8.05

1. Add Student
2. Display Students
3. Search Student
4. Update GPA
5. Delete Student
6. Exit
Choose an option: 2

Student Records:
ID: 123 | Name: harry | Age: 20 | GPA: 9.80
ID: 124 | Name: henry | Age: 17 | GPA: 8.05

1. Add Student
2. Display Students
3. Search Student
4. Update GPA
5. Delete Student
6. Exit
Choose an option: 3
Enter student ID to search: 128
Student not found
```


C:\Users\KJSCE\Desktop\exp

```
Choose an option: 3
Enter student ID to search: 128
Student not found.

1. Add Student
2. Display Students
3. Search Student
4. Update GPA
5. Delete Student
6. Exit
Choose an option: 4
Enter student ID to update GPA: 124
Enter new GPA: 9.3
GPA updated for student ID 124.

1. Add Student
2. Display Students
3. Search Student
4. Update GPA
5. Delete Student
6. Exit
Choose an option: 5
Enter student ID to delete: 1
Student not found.

1. Add Student
2. Display Students
3. Search Student
4. Update GPA
5. Delete Student
6. Exit
Choose an option: 5
Enter student ID to delete: 123
Student with ID 123 deleted successfully.

1. Add Student
2. Display Students
3. Search Student
4. Update GPA
5. Delete Student
6. Exit
Choose an option: 6
Program exited successfully!

Process returned 0 (0x0)   execution time : 48.385 s
Press any key to continue.
```

Post Lab Subjective/Objective type Questions:

1. What is a pointer in C?
   a) A variable that stores the value of another variable
   b) A variable that stores the memory address of another variable
   c) A function that points to another function
   d) A data type used for storing strings

Ans: b) A variable that stores the memory address of another variable

2. Which operator is used to get the memory address of a variable?
   a) `*` b) `&`
   c) `->` d) `#`

Ans: b) `&`

3. What does the `*` operator do when used with a pointer?
   a) Returns the memory address of the pointer
   b) Dereferences the pointer and accesses the value at the memory address
   c) Allocates memory for the pointer
   d) Frees the memory allocated to the pointer

Ans: b) Dereferences the pointer and accesses the value at the memory address

4. Which function is used to dynamically allocate memory in C?
   a) `malloc()` b) `calloc()`
   c) `realloc()` d) All of the above

Ans: d) All of the above

5. What is the difference between `malloc()` and `calloc()`?
   a) `malloc()` initializes memory to zero, while `calloc()` does not
   b) `calloc()` initializes memory to zero, while `malloc()` does not
   c) `malloc()` allocates memory for arrays, while `calloc()` allocates memory for single variables
   d) There is no difference

Ans: a) `malloc()` initializes memory to zero, while `calloc()` does not

6. What is the purpose of the `free()` function?
   a) To allocate memory b) To deallocate memory
   c) To resize memory d) To initialize memory

Ans: b) To deallocate memory

7. What happens if you dereference a null pointer?
   a) The program runs normally
   b) The program crashes or causes undefined behavior

c) The pointer is automatically allocated memory
d) The pointer points to the first memory location

Ans: b) The program crashes or causes undefined behavior

8. What is the output of the following code?

```
int x = 10;
int *ptr = &x;
printf("%d", *ptr);
```

a) 10
b) Memory address of `x`
c) Garbage value
d) Compilation error

Ans: a) 10

9. What is the purpose of the `realloc()` function?
a) To allocate new memory
b) To deallocate memory
c) To resize previously allocated memory
d) To initialize memory to zero

Ans: c) To resize previously allocated memory

10. What is a memory leak in C?
a) When memory is allocated but not used
b) When memory is allocated but not deallocated, causing wasted memory
c) When memory is deallocated twice
d) When memory is allocated using `calloc()` ‘

Ans: b) When memory is allocated but not deallocated, causing wasted memory

11. What is the output of the following code?

```
 int *ptr = (int *)malloc(sizeof(int));
*ptr = 5;
free(ptr);
printf("%d", *ptr);
```

a) 5 b) 0
c) Garbage value d) Runtime error

Ans: c) Garbage value

12. Which of the following is true about dynamic memory allocation?
a) Memory is allocated at compile time

b) Memory is allocated on the stack
c) Memory is allocated on the heap
d) Memory is automatically deallocated when the program ends
Ans. c) Memory is allocated on the heap

13. What is the correct way to allocate memory for an array of 10 integers dynamically?
a) `int arr[10];`
b) `int *arr = malloc(10);`
c) `int *arr = (int *)malloc(10 * sizeof(int));`
d) `int *arr = (int *)calloc(10, sizeof(int));`
Ans. c `int *arr = (int *)malloc(10* sizeof(int));`

14. What is a dangling pointer?
a) A pointer that points to a memory location that has been deallocated
b) A pointer that points to a null memory location
c) A pointer that points to an uninitialized memory location
d) A pointer that points to a valid memory location
Ans.a) A pointer that points to a memory location that has been deallocated

15. What is the output of the following code?
```
int *ptr = NULL;
printf("%d", *ptr);
```
a) 0 b) Garbage value
c) Runtime error d) Compilation error
Ans. c) Runtime error

Conclusion:

This experiment was successful in equipping me with knowledge of memory handling in C programming. I learnt how to allocate, reallocate, and deallocate memory and directly engage with my computer’s memory when dealing with data.

Signature of faculty in-charge with Date: