1.malloc for single integer

```
#include <stdio.h>
#include <stdlib.h>
int main() {
   // Dynamically allocate memory for a single integer
   int *ptr = (int*) malloc(sizeof(int));
   // Check if the memory was successfully allocated
   if (ptr == NULL) {
      printf("Memory allocation failed.\n");
      return 1;
   }
   // Assign a value to the dynamically allocated integer
   *ptr = 42;
   // Print the value
   printf("The value stored at ptr: %d\n", *ptr);
   // Free the allocated memory
   free(ptr);
   return 0;
}
```

2.malloc for an array

```
#include <stdio.h>
#include <stdlib.h>
int main()
 {
   int *ptr;
   int n = 5;
   ptr = (int*) malloc(n * sizeof(int));
   if (ptr == NULL) {
      printf("Memory not allocated.\n");
      return 1;  // exit program if memory allocation failed
   }
   printf("Memory successfully allocated using malloc.\n");
    for (int i = 0; i < n; i++)
    {
      ptr[i] = i + 1;
      printf("%d ", ptr[i]);
   }
   free(ptr);
   return 0;
}
```

3.calloc: contiguous allocation. It allocates memory for an array of elements, initializing all the bytes in the memory block to zero.

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
   int *arr;
   int n = 5;
   // Allocate memory for 5 integers and initialize all to zero
   arr = (int*) calloc(n, sizeof(int));
   // Check if memory allocation was successful
   if (arr == NULL) {
```

```
    printf("Memory allocation failed!\n");
    return 1;
  }
  // Print the array values (all should be zero initially)
  for (int i = 0; i < n; i++) {
    printf("%d ", arr[i]);
  }
  // Free the allocated memory
  free(arr);
  return 0;
}
```

5.realloc: The realloc() function in C is used to resize a previously allocated block of memory, either to expand or shrink it. void* realloc(void* ptr, size_t new_size);

```
#include <stdio.h>
#include <stdlib.h>
int main() {
  int *ptr;
  int n = 5, new_n = 10;
  // Step 1: Allocate memory for 5 integers using malloc
  ptr = (int*) malloc(n * sizeof(int));
  // Check if malloc was successful
  if (ptr == NULL) {
    printf("Memory allocation failed.\n");
    return 1;
  }
  // Assign values to the initial array
  for (int i = 0; i < n; i++) {
    ptr[i] = i + 1;
  }
  printf("Original array: ");
  for (int i = 0; i < n; i++) {
    printf("%d ", ptr[i]);
  }
  // Step 2: Resize the allocated memory to hold 10 integers
  ptr = (int*) realloc(ptr, new_n * sizeof(int));
  if (ptr == NULL) {
    printf("Memory reallocation failed.\n");
    return 1;
  }
for (int i = n; i < new_n; i++) {
    ptr[i] = i + 1;
  }
printf("\nResized array: ");
  for (int i = 0; i < new_n; i++) {
    printf("%d ", ptr[i]);
  }
  // Step 3: Free the dynamically allocated memory
  free(ptr);
  return 0;
}
```

6.void pointer

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

int main() {
   int a = 10;
   float b = 5.5;
   char c = 'A';

   // Declare void pointers
   void *ptr;
   // Point to an integer
   ptr = &a;
   printf("Value of integer a = %d\n", *(int*)ptr);  // Cast to int* before dereferencing

   // Point to a float
   ptr = &b;
   printf("Value of float b = %.2f\n", *(float*)ptr);  // Cast to float* before dereferencing
   // Point to a char
   ptr = &c;
   printf("Value of char c = %c\n", *(char*)ptr);  // Cast to char* before dereferencing
   return 0;
}
```

7.Lab Program 1

Design, Develop, and Implement a menu-driven Program in C for the following array operations.

a) Creating an array of N Integer Elements

b) Display of array Elements with Suitable Headings

c) Inserting an Element (ELEM) at a given valid Position (POS)

d) Deleting an Element at a given valid Position (POS)

```
#include <stdio.h>
#define MAX 100  // Maximum size of the array
// Function prototypes
void createArray(int arr[], int n);
void displayArray(int arr[], int n);
void insertElement(int arr[], int *n, int elem, int pos);
void deleteElement(int arr[], int *n, int pos);
int main() {
   int arr[MAX];  // Array to store elements
   int n = 0;     // Number of elements in the array
   int choice, elem, pos;
   while (1) {
      // Menu for array operations
      printf("\nMenu:");
      printf("\n1. Create an array of N elements");
      printf("\n2. Display array elements");
      printf("\n3. Insert an element at a given position");
      printf("\n4. Delete an element at a given position");
      printf("\n5. Exit");
      printf("\nEnter your choice (1-5): ");
      scanf("%d", &choice);
      switch (choice) {
```

```
            case 1:
                printf("Enter the number of elements in the array: ");
                scanf("%d", &n);
                if (n > MAX) {
                    printf("The maximum allowed elements are %d.\n", MAX);
                    n = 0;  // Reset array size if invalid input
                } else {
                    createArray(arr, n);
                }
                break;
            case 2:
                displayArray(arr, n);
                break;
            case 3:
                printf("Enter the element to insert: ");
                scanf("%d", &elem);
                printf("Enter the position to insert the element: ");
                scanf("%d", &pos);
                insertElement(arr, &n, elem, pos);
                break;
            case 4:
                printf("Enter the position to delete the element: ");
                scanf("%d", &pos);
                deleteElement(arr, &n, pos);
                break;
            case 5:
                printf("Exiting program...\n");
                return 0;
            default:
                printf("Invalid choice! Please choose between 1 and 5.\n");
        }
    }
    return 0;
}
void createArray(int arr[], int n) {
    int i;
    printf("Enter %d elements:\n", n);
    for (i = 0; i < n; i++) {
        scanf("%d", &arr[i]);
    }
    printf("Array created successfully.\n");
}
void displayArray(int arr[], int n) {
    int i;
    if (n == 0) {
        printf("Array is empty.\n");
    } else {
        printf("Array elements are: ");
        for (i = 0; i < n; i++) {
            printf("%d ", arr[i]);
        }
        printf("\n");
```

```
    }
}
void insertElement(int arr[], int *n, int elem, int pos) {
    int i;
    if (*n == MAX) {
        printf("Array is full! Cannot insert more elements.\n");
    } else if (pos < 1 || pos > *n + 1) {
        printf("Invalid position! Enter a position between 1 and %d\n", *n + 1);
    } else {
        // Shift elements to the right
        for (i = *n; i >= pos; i--) {
            arr[i] = arr[i - 1];
        }
        arr[pos - 1] = elem;
        (*n)++;  // Increase the size of the array
        printf("Element inserted successfully.\n");
    }
}
void deleteElement(int arr[], int *n, int pos) {
    int i;
    if (*n == 0) {
        printf("Array is empty! Nothing to delete.\n");
    } else if (pos < 1 || pos > *n) {
        printf("Invalid position! Enter a position between 1 and %d\n", *n);
    } else {
        for (i = pos - 1; i < *n - 1; i++) {   // Shift elements to the left
            arr[i] = arr[i + 1];
        }
        (*n)--;  // Decrease the size of the array
        printf("Element deleted successfully.\n");
    }
}
```