//Memory management - Dynamic memory allocation

//malloc() --- ptr = (cast-type*) malloc(byte-size)

//"malloc" or "memory allocation" method in C is used to dynamically allocate a single large block of memory with the specified size.
//It returns a pointer of type void which can be cast into a pointer of any form.

//calloc() --- ptr = (cast-type*)calloc(n, element-size);
//"calloc" or "contiguous allocation" method in C is used to dynamically allocate the specified number of blocks of memory of the specified type.

//free() --- free(ptr);
// "free" method in C is used to dynamically de-allocate the memory.

//realloc() --- ptr = realloc(ptr, newSize); where ptr is reallocated with new size 'newSize'.
// "realloc" or "re-allocation" method in C is used to dynamically change the memory allocation of a previously allocated memory.

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
        int* ptr;
        int n, i;

        printf("Enter number of elements:");
        scanf("%d",&n);
        printf("Entered number of elements: %d\n", n);
        //                      // Dynamically allocate memory using malloc()
        ptr = (int*)malloc(n * sizeof(int));
        if (ptr == NULL) {
           printf("Memory not allocated.\n");
           exit(0);
         }
        else {
                // Memory has been successfully allocated
                printf("Memory successfully allocated using MALLOC.\n");

                printf("The elements of the array are before initialization: ");
                     for (i = 0; i < n; ++i) {
                                        printf("%d, ", ptr[i]);
                                                }


                for (i = 0; i < n; ++i) {
                     ptr[i] = i + 1;
```

```
            }
            printf("The elements of the array after initialiation are: ");
            for (i = 0; i < n; ++i) {
                printf("%d, ", ptr[i]);
             }
      }
      free(ptr); // de-allocate memory

     printf("\n\nThe elements of the array after relasing ptr to allocated memory are: ");
     for (i = 0; i < n; i++) { printf("%d, ", ptr[i]);}
      printf("\n\n");

      ptr = (int*)calloc(n, sizeof(int)); // Dynamically allocate memory using calloc()

      if (ptr == NULL) {
         printf("Memory not allocated.\n");
         exit(0);
       }
      else {
           // Memory has been successfully allocated
            printf("Memory successfully allocated using CALLOC.\n");
         printf("The default elements of the array are: ");
                 for (i = 0; i < n; ++i) {printf("%d, ", ptr[i]); }

                  for (i = 0; i < n; ++i) { ptr[i] = i + 10; }

          printf("The elements of the array after update are: ");
            for (i = 0; i < n; ++i) {printf("%d, ", ptr[i]); }

          free(ptr); //de-allocate memory


           printf("\n\nThe elements of the array after releasing ptr to allocated memory are: ");
           for(i = 0; i < n; i++) { printf("%d, ", ptr[i]);}

           printf("\n\n");
      }
// re-allocation

         printf("\nEnter number of elements:");
         scanf("%d",&n);
         printf("Entered number of elements: %d\n", n); // Dynamically allocate memory using malloc()

         ptr = (int*)malloc(n * sizeof(int));
         if (ptr == NULL) { printf("Memory not allocated.\n"); exit(0);  }

         for (i = 0; i < n; ++i) { ptr[i] = i + 10; }
          printf("\nThe elements of the array before re-allocation are: ");
```

```
                        for (i = 0; i < n; ++i) {printf("%d, ", ptr[i]); }

            printf("\n\nEnter the new size of the array:\n");   // Dynamically re-allocate memory using
realloc()
            scanf("%d",&n);
            ptr = (int*)realloc(ptr, n * sizeof(int));
            if (ptr == NULL) { printf("Reallocation Failed\n"); exit(0);   }

           for (i = 0; i < n; ++i) { ptr[i] = ptr[i] + 10; }
              printf("\nThe elements of the array after re-allocation are: ");
                  for (i = 0; i < n; ++i) {printf("%d, ", ptr[i]); }
}
```