Answer key SlipNo. 1:

1.Write a C program to demonstrate the use of the atexit() function, which allows the registration of functions to be executed at program termination. The program should define multiple exit functions and register them in a specific order. Upon termination, the functions should execute in reverse order of their registration

1.Answers

```
#include <stdio.h>  // Include standard input-output header file
#include <stdlib.h> // Include standard library header file for atexit()

// Function called first at program termination
void first_exit_function() {
   printf("First exit function called\n");
}

// Function called second at program termination
void second_exit_function() {
   printf("Second exit function called\n");
}

// Function called third at program termination
void third_exit_function() {
   printf("Third exit function called\n");
}

int main() {
   // Register functions to be executed at program termination
   atexit(first_exit_function);  // Registers the first_exit_function
   atexit(second_exit_function); // Registers the second_exit_function
   atexit(third_exit_function);  // Registers the third_exit_function

   printf("Main function is running\n");
   return 0; // Program ends successfully
}

/*
Use of atexit():
1. The `atexit()` function allows registering functions to be called automatically at program termination.
2. These functions are executed in the reverse order of their registration, i.e., Last In First Out (LIFO).
3. This feature is useful for cleanup tasks, such as releasing resources or saving states, before the program
exits.
*/
```

2. Complete the C program to demonstrate malloc, calloc, realloc, and free for dynamic memory management of an integer array. Ensure proper allocation, resizing, initialization, and memory deallocation.

2.**Answer**

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

```c
#include <stdlib.h>

int main() {
   int *arr;
   int n;

   printf("Enter the number of elements: ");
   scanf("%d", &n);

   // Allocate memory using malloc
   arr = (int *)malloc(n * sizeof(int)); // Allocate memory for n integers
   if (arr == NULL) {                  // Check if pointer is null
      printf("Failed to allocate memory using malloc!\n"); // Display error message
      return 1;
   }

   printf("Memory allocated using malloc. Enter %d elements:\n", n);
   for (int i = 0; i < n; i++) {
      scanf("%d", &arr[i]); // Accept the number of elements dynamically
   }

   printf("Array elements after malloc:\n");
   for (int i = 0; i < n; i++) {
      printf("%d ", arr[i]); // Display all elements in the array
   }
   printf("\n");

   free(arr); // Free previous memory

   // Allocate memory using calloc and initialize to 0
   arr = (int *)calloc(n, sizeof(int));
   if (arr == NULL) {
      printf("Memory allocation using calloc failed!\n");
      return 1;
   }

   printf("Memory allocated using calloc. All elements initialized to 0:\n");
   for (int i = 0; i < n; i++) {
      printf("%d ", arr[i]); // Display array elements (all should be 0)
   }
   printf("\n");

   // Resize memory block
   printf("Enter the new size of the array: ");
   scanf("%d", &n);

   arr = (int *)realloc(arr, n * sizeof(int)); // Resize memory block using realloc
   if (arr == NULL) {
      printf("Memory reallocation failed!\n");
      return 1;
   }

   printf("Memory reallocated using realloc. Enter %d more elements:\n", n);
```

```
    for (int i = 0; i < n; i++) {
        arr[i] = i + 1; // Assign new values (e.g., 1, 2, 3, ...)
    }

    printf("Array elements after realloc:\n");
    for (int i = 0; i < n; i++) {
        printf("%d ", arr[i]); // Display all elements in the array
    }
    printf("\n");

    free(arr); // Free the allocated memory
    printf("Memory freed.\n");

    return 0;
}
```

3. Define a hypervisor and explain its role in virtualization. Differentiate between the types of hypervisors with suitable examples.

Answer

A **hypervisor**, also known as a **Virtual Machine Monitor (VMM)**, is software, firmware, or hardware that creates and runs virtual machines (VMs). It allows multiple operating systems to share a single physical hardware host by abstracting the hardware and providing isolated environments for each VM.s

Role of a Hypervisor in Virtualization:

1. **Resource Management**: It allocates CPU, memory, storage, and other resources among VMs efficiently.
2. **Isolation**: Provides a secure and isolated environment for each VM to prevent interference between them.
3. **Hardware Abstraction**: Allows different operating systems to run concurrently on the same hardware without conflicts.
4. **Flexibility**: Enables dynamic resource scaling and migration of VMs across physical hosts for load balancing and fault tolerance.

Types of Hypervisors:

1. **Type 1 (Bare-Metal Hypervisors)**:
   - Installed directly on physical hardware.
   - Does not require a host operating system, as it manages hardware directly.
   - **Examples**:
     - VMware ESXi
     - Microsoft Hyper-V
     - Xen Server
2. **Type 2 (Hosted Hypervisors)**:
   - Runs on top of an existing operating system.
   - Relies on the host OS for resource management and hardware interaction.
   - **Examples**:
     - VMware Workstation
     - Oracle VirtualBox
     - Parallels Desktop