Assignment No - 19

Q.1) What is the difference between malloc() and new, free and delete?

→

new()	malloc()
- used only in C++	used in C/C++
- Properly typed pointer	used void* (requires typecasting)
- Initializes objects and call constructors	Uninitialized memory
- for memory deallocation delete() used	for memory deallocation free() used
- Allocates and initializes an array of objects	Allocates a raw block of memory
- Exception handling is Throws, if memory allocation fails	Exception handling is Returns, if NULL on failure.

free()	delete
- used in C and C++	used in C++ only.
- Deallocates memory allocated by malloc(), calloc(), or realloc().	Deallocates memory allocated by new
- Does not call destructor	Calls destructors

Does not call destructor	Calls destructor
- Not type safe (works with void*)	Type safe (knows the object type)
- No distinction for arrays	Requires delete[] for arrays
- Cannot be overloaded	can be overloaded.
- Slightly faster	Slightly slower
- safe to call free(NULL)	safe to call delete NULL

Q.2] Write a program which is used to allocate memory for 10 integers using malloc()

→
```
#includude <stdio.h>
#include <stdlib.h>
int main()
{ int *arr;
  // allocating memory for 10 int using malloc
  arr = (int *)malloc(10 * sizeof(int));

  // check if memory allocation was successful
  if (arr = NULL){
   printf("Memory allocation failed!\n");
   return 1; }

  // Initialize and display the array
  printf("Enter 10 integers:\n");
```

```
for (int i = 0; i < 10; i++) {
    printf("%d", arr[i]); }
    printf("\n");

    // free the allocated memory
    free(arr);
return 0;
}
```

Q5) What is mean by dangling pointer.

A dangling pointer is a pointer that refers to a memory location that has been freed or deleted, leading to undefined behaviour if accessed.

A dangling pointer is a pointer that still holds the address of memory that has been deallocated or is out of scope, leading to undefined behaviour if accessed.

- How Dangling Pointers occurs

1] Using a pointer After free().
2] Using a pointer After delete
3] Returning address of a local variable

- How to avoid Dangling pointers.

- Set the pointer to NULL/nullptr after freeing memory.
- Use smart pointer in C++.
- Avoid returning address of local variables.

Q5] What is the return value of malloc() if memory manager is unable to allocate the memory?

→ If memory manager is unable to allocate memory malloc() return NULL.
This happens when there is not enogh memory available in the heap.
- malloc() returns NULL when allocation fails.
- Always check for NULL before using allocated memory.
- Avoid using uninitialized pointer to prevent segmentation faults.

Q6] Write a syntax which is used to allocate the memory for N floats dynamically using malloc(), Accept the value of N from user at run time.

→ float *ptr = (float *) malloc (N * (sizeof (float));

malloc(N * sizeof (float)) - dynamically allocates memory for N floats.

Q7] Allocate dynamic memory for array of 5 element where each elements is of below structure type

```
struc hello {
    float f;
    int i;
};
```

→

```c
#include <stdio.h>
#include <stdlib.h>
struct hello {
    float f;
    int i;
};

int main()
{  // Declaring pointer to struct demo hello
    struct hello *ptr;

    // Allocating memory for 5 elements dynamically
    ptr = (struct hello *)malloc(5 * sizeof(struct hello));

    // Accepting values from the user
    printf("Enter values for 5 elements :\n");
    for (int i = 0; i < 5; i++)
    { printf("Element %d:\n", i+1);
      printf("Enter float value:");
      scanf("%f", &ptr[i].f);
      printf("Enter int value:");
      scanf("%d", &ptr[i].i);
}
// Displaying stored value.
    printf("\nEntered values:\n");
    for (int i = 0; i<5; i++){
      printf("Element %d -> float: %.2f, Int: %d\n",
                i+1, ptr[i].f, ptr[i].i);
}

    // freeing allocated memory
    free(ptr);
return 0;
}
```

Q8] Explain the internal working of new operator.

→ The new operator in C++ is used to dynamically allocate memory from the heap. it perform following operations:

1) Memory allocation (using malloc() or similar).
2) Constructor call (for objects).
3) Returns a pointer.

eg. int *ptr = new int(5);

- calls memory allocation function (malloc() or operator new()) to allocate memory for an int.
- If allocation fails, it throws a std::bad_alloc exception.
- If allocation succeeds, stores 5 in the allocated memory.
- Returns a pointer to the allocated memory.

Q9] What is the need of typecasting for the return value of malloc().

→ The function malloc() returns a void pointer, which is a generic pointer type that can be assigned to any other pointer type without typecasting in C. However, in C++ typecasting is mandatory.

Q10] Explain the working of below application, draw its diagramatic representation.

```
#include<stdio.h>
#include <stdlib.h>
int main()
{  int size = 0;
   int *p = NULL;
   int iCnt = 0;
```

```
printf("Enter number of elements \n");
scanf("%d", &size);

p = (int *) malloc (size * sizeof(int));

printf("Please enter the elements \n");
for(iCnt = 0; iCnt < size; iCnt++)
{ scanf("%d", &p[iCnt]);
}

printf("entered elements are \n");
for (iCnt = 0; iCnt < size; iCnt++)
{ scanf("%d", p[iCnt]);
}

free(p);
return 0;
}
```

→ The given c program dynamically allocates memory for an integer array. It takes user input for its elements and printfs the entered values.

malloc(size * sizeof(int))

It allocates memory for size integers.

The allocated memory is assigned to pointer P.

The program uses a for loop for take size integer inputs and stores them in dynamically allocated memory.

Another for loop interates through the allocated memory and paints each stored value.

free(p) is called to release the allocated memory preventing memory leaks.