Assignment No- 18

Q1] Explain the difference between static and dynamic memory allocation.

→

Static Memory Allocation	Dynamic Memory Allocation.
- There is no specific function.	Have to use specific fun' for allocation and deallocat
- Calculation for memory allocation done at compile time.	Calculation for memory allocation done at run time.
- It may lead problem for storage or waste of memory	Ther is no such a problem of shortage or wastage.
- we can't deallocate memory after its use get Completed.	we can deallocate memory after use get completed.
- There is no memory allocation failure at run time	There is no such memory allocation failure at run time.
- It is fast way of memory allocation as there is no calculation at run time	It is a slowest way.
- Memory get allocated based on there storage (either stack, data, cpu).	Memory get allocated inside heap section.

Q2] What are the advantages and dis advantages of dynamic memory allocation over static memory allocation.

→ Advantages

1] Efficient Memory Utilization
Dynamic memory allocation allows programs to use memory as needed.

2] Flexibility
we can allocate memory at runtime, enabling the program to handle varying amounts of data.

3] Memory Size Scalability
The size of dynamically allocated memory can be adjusted during runtime.

4] Lifetime Management.
Dynamically allocated memory can exist beyond the scope of a function if needed.

5] Support for Data Structures
Dynamic memory is essential for implementing advanced data structures like linked lists, trees, and graphs, which require memory to grow or shrink dynamically.

Disadvantages-

1] Overhead
Dynamic memory allocation is slower compared to static allocation

2] Memory fragmentation
over time, the heap can become fragmented due to repeated allocation and deallocation, which can lead to inefficient use of memory.

3) Manual Management
Dynamically allocated memory requires explicit deallocation. If memory is not freed, it can lead to memory leaks.
4) Complexity
Managing dynamically allocated memory increase program complexity, as you must ensure proper allocation and deallocation.
5) Risk of Errors.
Common errors like double freeing, accessing freed memory and out-of-bound access are more likely in dynamic memory allocation.

3] List down the functions and its syntaxes which are used in C.

→ • Input/Output function.
1] Printf - prints formatted output to the console.
Syntax- printf("format string", arguments);

2] Scanf - Reads formatted input from the console
Syntax- Scanf("format string", &variables);

3] gets - Reads a string until a newline character is encountered
Syntax - gets(string);

4] puts - outputs a string followed by a newline
Syntax- puts(string);

• Memory Management Functions <stdlib.h>

2) Manual Management
Dynamically allocated memory requires explicit deallocation. If memory is not freed, it can lead to memory leaks.
4) Complexity
Managing dynamically allocated memory increase program complexity, as you must ensure proper allocation and deallocation.
5) Risk of Errors
Common errors like double freeing, accessing freed memory and out-of-bound access are more likely in dynamic memory allocation.

3] List down the functions and its syntaxes which are used in C.

→ • Input/output function
1) Printf - prints formatted output to the console.
Syntax- printf("format string", arguments);

2) Scanf - Reads formatted input from the console.
Syntax- Scanf("format string", &variables);

3) gets - Reads a string until a newline character is encountered
Syntax- gets(string);

4) puts - outputs a string followed by a newline
Syntax- puts(string);

• Memory Management functions <stdlib.h>

③ Manual Management
Dynamically allocated memory requires explicit deallocation. If memory is not freed, it can lead to memory leaks.

④ Complexity
Managing dynamically allocated memory increase program complexity, as you must ensure proper allocation and deallocation.

⑤ Risk of Errors.
Common errors like double freeing, accessing freed memory and out-of-bound access are more likely in dynamic memory allocation.

3] List down the functions and its syntaxes which are used in C.

→ • Input/output function.

① Printf - prints formatted output to the console.
Syntax- printf(" format string", arguments);

② scanf - Reads formatted input from the console
Syntax- scanf("format string", &variables);

③ gets - Reads a string until a newline character is encountered
Syntax - gets(string);

④ puts - outputs a string followed by a newline
Syntax- puts(string);

• Memory Management Functions <stdlib.h>

1] malloc - allocates memory dynamically
void *malloc (size_t size);

2] calloc - Allocates memory for an array and initializes all elements to zero
void *calloc (size_t num, size_t size);

3] realloc - Reallocates memory to resize a previously allocated block.
void *realloc (void *ptr, size_t size);

4] free - frees dynamically allocated memory.
void free (void *ptr);

Q4] Which functions / operators are used in c++ for dynamic memory allocation and deallocation.

→ Operators for dynamic Memory Allocation.

1] new - Allocates memory dynamically for a single object or an array of objects.
Syntax- int *ptr = new int;

2] delete - Deallocates memory allocated with new
Syntax- delete ptr;

Functions from C (also usable in C++)

1] malloc: Allocates memory dynamically without calling a constructor.

2] calloc - Allocates memory for an array and initializes all elements to zero.

3] realloc - Resizes previously allocated memory

4] free - Deallocates memory allocated with malloc, calloc, realloc.

Q5] Write down the difference malloc() and calloc()

→

Malloc ()	Calloc ()
Memory Allocation	Contiguous Allocation
1] Malloc() is a function that creates one block of memory of a fixed size.	calloc() is a function that assigns a specified number of blocks of memory to a single variable.
2] malloc() only takes one argument	calloc() takes two arguments.
3] It is faster than calloc	It is slower than malloc()
4] Used to indicate memory allocation	Used to indicate contiguous memory allocation.
5] It does not initialize the memory to zero	It initializes the memory to zero.
6] Syntax- void *malloc(size_t size)	Syntax void * calloc(size_t num, size_t size);

Q.] Explain the prototype of malloc function with example.

→ Prototype -

void * malloc (int size)

e.g.

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
    int Arr[5];    // static memory allocation
    float fvalue;
    double Arr[4];

    int isize = 0;
    int *ptr = NULL;
    printf("Enter size of array \n");
    scanf("%d", &isize);
    ptr = (int *)malloc(isize * sizeof(int));

    ptr[0] = 10;
    ptr[1] = 20;
    ptr[2] = 30;
    printf("ptr -> [1] %d", ptr[1]);

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

Q6] Explain the prototype of malloc function with example.

→ Prototype -

void * malloc (int size)

e.g.

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
    int Arr [5];    // static memory allocation
    float fvalue;
    double Brr [4];

    int isize = 0;
    int *ptr = NULL;
    printf ("Enter size of array \n");
    scanf ("%d", &isize);
    ptr = (int *)malloc (isize * sizeof (int));

    ptr [0] = 10;
    ptr [1] = 20;
    ptr [2] = 30;
    printf (" ptr → [1] %d", ptr [1]);

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

Q7] Why return value of malloc(), calloc() and realloc() is void*?

→ Because,

• void* is a generic pointer
- A void* can be typecast to any other pointer
- This allows flexibility, so the functions can allocate memory for any data type

• Avoids Type Restrictions
- If malloc(), calloc(), realloc() returned a specific type, it would limit the usage to only int.
- Since void* is type-~~casting~~ agnostic, these functions can be used for any data structure

• Explicit Type Casting in C++.
In C, implicit conversion from void* to another pointer type is allowed, so casting is not required
- In C++, implicit Conversion from void* to another pointer type is not allowed, so an explicit cast is required.

Q8] What are the different use of realloc()?

→ 1] Expanding Memory Block (Increasing Size)
If initially allocated memory is insufficient, realloc() can be used to allocate more space while keeping the existing data.
- No need to manually copy old data, realloc() handles it.

2] Shrinking Memory Block (Reducing Size)
realloc() is useful when reading an unknown

number of inputs
- Avoids allocating unnecessary large buffers

Q9] What will happen if we use first parameter of realloc() as NULL?

→ If first parameter of realloc() is NULL the function behaves like malloc().
It allocates new memory of the requested size and returns a pointer to it.
• Behaviour of realloc(NULL, size)
- Equivalent to malloc(size), meaning allocates new memory instead of resizing.
- The contents of the newly allocated memory are uninitialized
• It is safe to use and prevents unnecessary if conditions for malloc(), and realloc().

Q10] What will happen if second parameter of realloc() is 0!

→ When calling realloc(ptr, 0) the behavior depends on the c standard and the compiler implementation
• Equivalent to free(ptr)
Most standard c libraries treat realloc(ptr, 0) as equivalent to free(ptr), realeasing the allocated memory.
- The allocated memory is freed, and arr become NULL.

number of inputs
- Avoids allocating unneccessary large buffers

Q9] What will happen if we use first parameter of realloc() as NULL?

→ If first parameter of realloc() is NULL the function behaves like malloc().
It allocates new memory of the requested size and returns a pointer to it.
• Behaviour of realloc (NULL, size)
- Equivalent to malloc (size), meaning allocates new memory instead of resizing.
- The contents of the newly allocated memory are uninitialized
- It is safe to use and prevents unnecessary if conditions for malloc(), and realloc().

Q10] What will happen if second parameter of realloc() is 0!

→ When calling realloc (ptr, 0) the behavior depends on the c standard and the compiler implementation
• Equivalent to free (ptr)
Most standard c. libraries treat realloc (ptr, 0) as equivalent to free (ptr), realeasing the allocated memory.
- The allocated memory is freed, and arr become NULL.