(Assignment - 1)

1. Explain the purpose of dynamic memory allocation in C Discuss the difference between stack memory and heap memory.

Dynamic memory allocation in C allows programs to allocate memory at runtime, rather than at compile time. This provides flexibility in managing memory resources and enables efficient memory usage. The purpose of dynamic memory allocation includes.

1. Variable Memory Requirements :

Programs often encounter situation where the required memory size is not known beforehand or varies during execution. Dynamic memory allocation allows programs to adapt to such scenarios by allocating memory as needed.

2. Dynamic Data Structures :

Dynamic memory allocation is essential for creating dynamic data structures such as linked lists, trees, and graphs, where the number of elements may change dynamically.

3. Resource Management :

It enables efficient memory management by allocating memory only when required and releasing it when no longer needed, preventing memory wastage.

Differences Between stack memory and heap memory.

Stack memory :	Heap memory :
1. Allocation and Deallocation	
Allocation and deallocated automatically by the compiler as functions are called and return.	Allocated and deallocated manually by the programs using functions like malloc(), calloc(), realloc() and free().

2. Scope	
Limited to the scope of the block or function in which they are declared.	can be accessed globally across the program, the depending on how it's allocated.
3. Size Limitations	
Limited in size and typically smaller than heap memory	Limited only by the available system memory and virtual memory.
4. Access speed :	
Faster access since it's organize as a stack and follows a Last In, First Out (LIFS) structure.	Slightly slower access due to dynamic allocation mechanisms.
5. Fragmentation: Less prone to fragmentation since memory is allocated and deallocated in a strict order	More prone to fragmentation due to dynamic allocation to and deallocation, which can lead to memory leaks if not managed properly.
6. Stack memory is used for local variables and function calls with a limited size and automatic management	Heap memory is used for Dynamic allocation of memory with flexible size and manual management by the programmer.

2) Compare malloc and calloc. what is the key difference between them? provide an example demonstrating their usage.

malloc() and calloc() are both functions used for dynamic memory allocation in C, but they have a key difference in how they initialize the allocated memory.

① malloc():

- The malloc() function allocates a block of memory of the specified size in bytes.
- It does not initialize the allocated memory, leaving its content uninitialized (garbage values).
- Syntex:

```
void *malloc(size_t size);
```

② calloc()

- The calloc() function allocates a block of memory for any array of elements, each of which has a size of size bytes.
- It initialize the allocated memory to zero.
- Syntex:

```
void* calloc(size_t num_element
             , size_t element_size);
```

Key Difference:

- The main difference between malloc() and calloc() is that calloc() initializes the allocated memory to zero, while malloc() does not.

Example:

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

int main() {
    int *arr1, *arr2;
    int size = 5;
    arr1= (int*) malloc(size * size of (int));
    if (arr1 == NULL) {
        printf("Memory allocation failed. \n");
        return -1;
    }
    print ("Memory allocated using malloc : \n");
    for (int i=0 ; i < size ; i++);
    { printf ("%d", arr[i]);
    }
    print ("\n");
arr2 = (int*) calloc (size , size of (int));
if (arr2 == NULL) {
        printf ("Memory allocation failed.\n");
        return -1;
    }
```

```
    printf("Memory allocated using calloc : \n");
    for (int i=0 ; i<size ; i++)
    {   printf("%d", arr[2]);
    }
    printf("\n");
    free (arr1);
    free (arr2);
    return 0;
}
```

3) Explain the purpose of realloc provide a scenario where resizing an existng memory block is necessary.

The purpose of the realloc function in C is to reallocate memory for a previously allocated block. it allow you to change the size of an existing memory block, either making it larger or smaller, while preserving the data it currently holds as much as possible.

One scenario where resizing to an existing memory block is necessary is when you're working with dynamic data structures like arrays, lists, or strings. For example, let's says you having have a dynamic array to store elements, and you initially allocate memory for 10 elements. As you add more elements, use you realize that you need more space to accommodate them. Instead of allocating a completely new array and copying all the elements from the old array to the new one. you can use realloc to resize the existing array, potentially saving time and resources.

④ what is a dangling pointer ? How can you avoid it?

A dangling pointer is a pointer that points to memory that has been deallocted, or freed, leading to undefined behaviour when dereferenced. This can occur when the memory block to which the pointer points is released, but the pointer is not updated to reflect this change.

* To avoid dangling pointers, you can follow these practises:

1) Nullify pointers: After deallocating memory, set the pointer to NULL (or nullptr in C) to prevent accidental dereferencing.

2) Avoid pointer Aliasing: Ensure that a pointer doesn't get assigned the address of another valid variable that is about to be deallocated.

3) Lifetime Management: Keep track of the lifetime of objects and their associated pointers. make sure that pointers are not used after the memory they point to has been deallocated.

⑤ Explain the difference between free and delete in C and C++. when should you use each.

In C, the free() function is used to deallocated memory that was previously allocated using functions like malloc(), calloc(), or realloc(), it's used exclusively allocated memory.

In C++, the delete operator is used to deallocate memory that was previously allocated using the new operator. Similarly, delete[] is used to deallocate memory that was allocated with new[], which is typically used for arrays.

The key difference are:

1) Syntax:- In C you use free(ptr) to deallocate memory, while in C++, you use delete ptr for single objects and delete[] ptr for arrays.

2) Type Awareness:-
In C, free() is not aware of the type of memory being deallocated; it simply frees the memory at the specified address. In C++ delete and delete[] are type-aware, meaning they call the appropriate destructors for objects before deallocating memory.

When to use each;

- In C, always use free() to deallocate memory allocated with malloc(), calloc(), or realloc()
- In C++ use delete to deallocate memory allocated with new and delete[] to deallocate memory allocated with new[].

6) what happens if you forget to free dynamically allocated memory? How can memory leaks impact program performances?

Forgetting to free dynamically allocated memory leads to memory leaks. Memory leaks occur when memory that is no longer needed is not deallocated causing it to remain allocated indefinetely. This can lead to a gradual depletion of ~~avail~~ available memory, eventually causing the program or System to run out of memory and crash. Memory leaks can impacts program performance by Showing down the System due to increased memory usage and potential causing instability or crashes. over time. if memory leaks accumulate, they can significantly degrade System performance and reliability.

7) Explain the concept of an array of pointers. How can you use it to store address of aynamically allocated memory block?

An array of pointers is an array where each element holds the memory address of another variable or data structure. This allows for flexibility in managing memory and accessing different data structure efficiently.

To use an array of pointer to store address of dynamically allocated memory blocks, you can follow these steps:

① Allocate memory dynamically for the array of pointers using functions like malloc or calloc

② Allocate memory for each individual block using functions like malloc or calloc, and store their addresses in the respective elements of the array.

③ Access or manipulate the dynamically allocated memory blocks using the array of pointers

⑧ what is the return type of malloc and calloc ? why is it void* ?

The return type of both malloc and calloc functions in C is void*. This means they return a pointer to void, which is a generic pointer type. The reason for this is to provide flexibility in memory allocation

Since malloc and calloc allocate memory dynamically, they don't know in advance what type of data will be store in the allocated memory. Therefore they return a generic pointer (void*) which can be implicitly converted to any other pointer type in C. This allows the programmer to cast the returned pointer to the desired data type when assigning it to a pointer variable

#For example:

```
int *ptr = (int*) malloc (size of (int)); // casting
                                          void* to int*
```

9) Explain the difference between Stack memory and heap memory in terms of storage duration, allocation, and deallocations.

Stack memory and heap memory are two distinct regions in a computer's memory that serve different purpose and have different characteristics

① Storage Duration:

Stack memory	Heap memory
Stack memory: Variables allocated on the stack have a short storage duration. They exist only with in the Scope of the function in which they are defined when the function exits, the memory allocated for stack variables is automatically deallocated	Heap memory: Memory allocated on the heap has a longer storage duration. it persists until explicitly deallocated by the programmer using functions like free() in C. memory on the heap can be allocated and deallocated dynamically during program execution.

② Allocation:

Stack memory: Memory allocation on the Stack is done automatically by the compiler when a function is called. Local variables, functions parameter, and return addresses are typically stored on the Stack	Heap memory: Memory allocation on the heap is done explicitly by the programmer using different functions like malloc(), calloc(), or new (in C/C++). This allows for dynamic memory allocation, where memory can be allocated based on runtime conditions.

③ Deallocation:

Stack memory: Deallocation of stack memory is automatic and handled by the compiler when the function exits. Variables are popped off the stack in a last-in-first-out (LIFO) manner.

Heap memory: Deallocation of heap memory is manual and the responsibility of the programmer. Failing to deallocate memory allocate on the heap can lead to memory leaks, where memory is not released even though it's no longer needed.