UNIT NO. 1

Dynamic Memory Allocation in C

- The concept of **dynamic memory allocation in c language** enables the C programmer to allocate memory at runtime. Dynamic memory allocation in c language is possible by 4 functions of stdlib.h header file.
- Dynamic memory allocation is a way for a computer program to request and use memory from the computer's memory system while the program is running. It allows programs to allocate memory as needed rather than pre-allocating a fixed amount of memory at the beginning.
- Think of it like asking for space in a parking lot as you need it. Instead of reserving a specific spot for your car ahead of time, you request a spot when you arrive, and you can request more spots if you need them. This flexibility helps programs use memory efficiently and adapt to different situations during runtime.
- Functions of DMA

1. malloc()
2. calloc()
3. realloc()
4. free()

Static Testing and dynamic Testing

Basis for Comparison	Static Memory Allocation	Dynamic Memory Allocation
Abbreviated as	SMA	DMA
Basic	Fixed type of memory- allocation.	Variable type of memory allocation.
Occurs	Befor program execution	During program execution
Space required	More	Comparatively less
Memory reusability	Not exist	Exist
Execution speed	Fast	Comparatively slow
Implementation	Simple	Complex
Efficiency	Less	More
Fragmentation issue	Not exist	Exist
Memory size	Unchangeable	Changeable

1] malloc () function is used to allocate space in memory during the execution of the program. It does not initialize the memory allocated during execution. It carries garbage value. it returns null pointer if it couldn't able to allocate requested amount of memory.

Syntax:

```
pointer=(type*) malloc (number *sizeof(type));
Ex:    (int*)malloc(20*sizeof(int));
        char *k;
       k = malloc( 20 * sizeof(char) );
```

Example of Malloc

```
#include <stdlib.h>
int main() {
  int n, i, *ptr, sum = 0;
  printf("Enter number of elements: ");
  scanf("%d", &n);
  ptr = (int*) malloc(n * sizeof(int));
  if(ptr == NULL) {
    printf("Error! memory not allocated.");
    exit(0);
  }
  printf("Enter elements: ");
  for(i = 0; i < n; ++i) {
    scanf("%d", ptr + i);
```

```
    sum += *(ptr + i);
  }
  printf("Sum = %d", sum);
  free(ptr);
  return 0;
}
```

2] calloc () :

- function is similar to malloc () function. calloc () initializes to default value.
- This is a function in C that allocates a block of memory for an array of elements. It stands for "contiguous allocation". When you use calloc, you tell the computer how many elements you need and how big each element is. Then, it gives you a chunk of memory big enough to hold all those elements, and it sets all the bits to zero.
- So, if you want, say, 10 integers worth of memory using calloc, it will give you a block of memory where you can store 10 integers, and it will make sure all those integers start with a value of 0.
- In simpler terms, calloc is like renting a bunch of boxes in a warehouse, and the nice thing is, when you get those boxes, they're all empty and ready for you to use!

Syntax: pointer=(type*)calloc(arraysize,datatype);
Ex: char *k;

```
k = calloc( 20, sizeof(char) );
```

Example of calloc and free

```
#include <stdlib.h>
int main() {
  int n, i, *ptr, sum = 0;
  printf("Enter number of elements: ");
  scanf("%d", &n);
 ptr = (int*) calloc(n, sizeof(int));
  if(ptr == NULL) {
    printf("Error! memory not allocated.");
    exit(0);
  } printf("Enter elements: ");
  for(i = 0; i < n; ++i) {
    scanf("%d", ptr + i);
    sum += *(ptr + i);
  } printf("Sum = %d", sum);
  free(ptr);
  return 0;
}
```

3] realloc ()

- function modifies the allocated memory size by malloc () and calloc () functions to new size. If enough space doesn't exist in memory of current block to extend, new

block is allocated for the full size of reallocation, then copies the existing data to new block and then frees the old block.

- This function in C helps you change the size of memory that you've already asked for using functions like malloc or calloc. Let's say you initially asked for space to store 10 items, but then you realize you need space for 15 items. Instead of asking for a completely new block of memory, you can use realloc to expand the existing block to accommodate the extra items.
- Similarly, if you find you have too much space allocated, you can use realloc to shrink the block of memory. It's like saying, "Hey, I rented this storage unit for 10 boxes, but now I only need space for 8 boxes. Can I give back some of this space?" And realloc handles that for you.
- In essence, realloc helps you manage your memory more efficiently by adjusting the size of the memory block you've previously allocated.

Example of Realloc

```
#include <stdlib.h>

int main() {

int *ptr, i , n1, n2;

printf("Enter size: ");

scanf("%d", &n1);

ptr = (int*) malloc(n1 * sizeof(int));

printf("Addresses of previously allocated memory:\n");

for(i = 0; i < n1; ++i)
```

```
printf("%pc\n",ptr + i);
printf("\nEnter the new size: ");
scanf("%d", &n2);
ptr = realloc(ptr, n2 * sizeof(int));
printf("Addresses of newly allocated memory:\n");
for(i = 0; i < n2; ++i)
printf("%pc\n", ptr + i);
free(ptr);
return 0;
}
```

Syntax:pointer= realloc(pointername,newsize);
Ex:k= realloc(k,40);

4] free () function frees the allocated memory by malloc (), calloc (), realloc () functions and returns the memory to the system.
Syntax: free(pointer);
Ex:free(k);

S.no	malloc()	calloc()
1	It allocates only single block of requested memory	It allocates multiple blocks of requested memory

2	int *ptr;ptr = malloc(20 * sizeof(int));For the above, 20*4 bytes of memory only allocated in one block. Total = 80 bytes	int *ptr;Ptr = calloc(20, 20 * sizeof(int));For the above, 20 blocks of memory will be created and each contains 20*4 bytes of memory. Total = 1600 bytes
3	malloc () doesn't initializes the allocated memory. It contains garbage values	calloc () initializes the allocated memory to zero
4	type cast must be done since this function returns void pointer int *ptr;ptr = (int*)malloc(sizeof(int)*20);	Same as malloc () function int *ptr;ptr = (int*)calloc(20, 20 * sizeof(int));

Pointers:

- Are one of the core components of the C programming language. A pointer can be used to store the **memory address** of other variables, functions, or even other pointers. The use of pointers allows low-level memory

access, dynamic memory allocation, and many other functionality in C.

int **pp=&p;

- Pointers in C are like signposts or arrows that show where something is stored in the computer's memory. Instead of holding the actual data, a pointer holds the memory address of where the data is stored.
- Declaration and Initialization: When you declare a pointer, you're telling the computer that this variable will hold the memory address of a specific type of data. For example, int *ptr; declares a pointer that can point to an integer. To make a pointer point to something, you assign it the memory address of another variable using the & operator. For example, ptr = &x;, where x is an integer variable.
- Dereferencing: To access the actual value stored at the memory address pointed to by a pointer, you dereference it using the * operator. For example, *ptr gives you the value stored at the memory address stored in ptr.
- Pointer Arithmetic: You can do arithmetic with pointers to move them around in memory. For example, adding 1 to a pointer moves it to the next memory location of the type it points to

Types of pointers:

There are many types of pointers:

Null Pointer: A pointer which is pointing to nothing is known as a null pointer. Null pointer points to the base address of the segment.

Wild pointer: A pointer in C language which has not been initialized is known as a wild pointer.

Dangling Pointer: A pointer which points to the memory address of any variable but later the data or variable gets deleted from that location while the pointer still points to such memory, such a pointer is known as a dangling pointer.

Near Pointer: A pointer which points only 64KB data segment or segment number 8 is known as near pointer.
Far pointer: A pointer which can point or access the whole residence memory of RAM or which can access all 16 segments is known as far pointer.

Generic pointer: A pointer which can point void type of data is known as a generic pointer. Void pointer or generic pointer is a special type of pointer that can be pointed at objects of any data type. A void pointer is declared like a normal pointer, using the void keyword as the pointer's type. Void pointer or generic pointer is a special type of pointer that can be pointed at objects of any data type. A void pointer is declared like a normal pointer, using the void keyword as the pointer's type

Pointer to pointer or double pointer: pointer variable stores the address of another pointer variable's address.
Syntax: datatype **variablename;
Ex: int m=40;
int *p=&m;
int **pp=&p;

Programming application of arrays of Pointers

- In Programming arrays and pointers simplifying data handling. Arrays names act as pointers to their starting points. Pointers help access data quickly.

- This collaboration allows for efficient manipulation of data within arrays. This concept also extends to multidimensional arrays and creating arrays of pointers to manage various data.
- Generally, pointers are the variables which contain the addresses of some other variables and with arrays a pointer stores the starting address of the array.
- Array name itself acts as a pointer to the first element of the array and also if a pointer variable stores the base address of an array then we can manipulate all the array elements using the pointer variable only.
- Pointers can be associated with the multidimensional arrays (2-D and 3-D arrays) as well. Also, We can create an array of pointers to store multiple addresses of different variables.
- Pointers and Array representations are very much related to each other and can be interchangeably used in the right context.
- An array name is generally treated as a pointer to the first element of the array and if we store the base address of the array in another pointer variable, then we can easily manipulate the array using pointer arithmetic in a C Progra

Array of Pointer

A pointer array is a homogeneous collection of indexed pointer variables that are references to a memory location. It is generally used in C Programming when we want to point at

multiple memory locations of a similar data type in our C program. We can access the data by dereferencing the pointer pointing to it.

Syntax:

pointer_type *array_name [array_size];

Here,

- **pointer_type:** Type of data the pointer is pointing to.
- **array_name:** Name of the array of pointers.
- **array_size:** Size of the array of pointers.

Example:

```
#include <stdio.h>

int main()

{

	int var1 = 10;

	int var2 = 20;

	int var3 = 30;

	int* ptr_arr[3] = { &var1, &var2, &var3 };

	for (int i = 0; i < 3; i++) {

		printf("Value of var%d: %d\tAddress: %p\n", i + 1, *ptr_arr[i], ptr_arr[i]);

	}


	return 0;

}
```

Array of Pointers to Character

One of the main applications of the array of pointers is to store multiple strings as an array of pointers to characters. Here, each pointer in the array is a character pointer that points to the first character of the string.

Syntax:

char *array_name [array_size];

After that, we can assign a string of any length to these pointers.

Example

```
char* arr[5] = { "kgf", "veer", "jaiho", "charleey", "itstimetodorock" }
```

```
#include <stdio.h>
int main()
{       char str[3][10] = { "kgf", "veer", "jaiho" };
        printf("String array Elements are:\n");
        for (int i = 0; i < 3; i++) {
        printf("%s\n", str[i]);
       }
  return 0;
}
```