**Dynamic Memory Allocation & File Handling in C**

## ****Introduction****

C provides powerful features for **dynamic memory allocation** and **file handling**, which allow efficient memory usage and persistent data storage. This lecture covers these topics with examples and diagrams for better understanding.

# ****Part 1: Dynamic Memory Allocation in C****

## ****Why Dynamic Memory Allocation?****

* In static memory allocation, memory is allocated at compile-time, leading to fixed-size arrays.
* Dynamic memory allocation allows allocating memory at **runtime**, making programs more flexible.

## ****Functions for Dynamic Memory Allocation****

C provides four functions for dynamic memory management:

### ****1. malloc()****

* Allocates memory but does **not initialize** it.
* Returns a void pointer, which must be typecasted.

**Example:**

#include <stdio.h>

#include <stdlib.h>

int main() {

int \*ptr = (int\*) malloc(5 \* sizeof(int));

if (ptr == NULL) {

printf("Memory allocation failed!\n");

return 1;

}

printf("Memory allocated successfully!\n");

free(ptr); // Freeing allocated memory

return 0;

}

### ****2. calloc()****

* Allocates and initializes memory with zero.
* Takes two parameters: **number of blocks** and **size of each block**.

**Example:**

int \*arr = (int\*) calloc(5, sizeof(int));

### ****3. realloc()****

* Resizes previously allocated memory.
* Preserves existing data while extending or reducing the memory block.

**Example:**

ptr = (int\*) realloc(ptr, 10 \* sizeof(int));

### ****4. free()****

* Deallocates memory to prevent memory leaks.

**Memory Allocation Diagram:**

Heap Memory

--------------------

| malloc / calloc |

| dynamically |

| allocated |

| memory |

--------------------

## ****Common Errors in Dynamic Memory Allocation****

1. **Memory Leaks**: Forgetting to free allocated memory.
2. **Dangling Pointers**: Using a pointer after freeing memory.
3. **Double Free**: Freeing the same memory twice.

# ****Part 2: File Handling in C****

## ****Why Use Files?****

* Data persists even after the program ends.
* Large data storage without memory limitations.

## ****File Handling Functions****

C uses the FILE structure to handle files.

### ****1. Opening a File****

* fopen(filename, mode): Opens a file in specified mode.
* Modes:
  + "r": Read
  + "w": Write (creates file if not exists, overwrites existing file)
  + "a": Append
  + "r+", "w+", "a+" (Read+Write modes)

**Example:**

FILE \*fp = fopen("data.txt", "w");

if (fp == NULL) {

printf("Error opening file!\n");

return 1;

}

fclose(fp);

### ****2. Writing to a File****

* fprintf(): Formatted writing.

**Example:**

FILE \*fp = fopen("data.txt", "w");

fprintf(fp, "Hello, World!\n");

fclose(fp);

### ****3. Reading from a File****

* fgetc(), fgets(): Character and string reading.

**Example:**

char buffer[100];

FILE \*fp = fopen("data.txt", "r");

fgets(buffer, 100, fp);

printf("Read: %s", buffer);

fclose(fp);

### ****4. Closing a File****

* Always close files using fclose() to free resources.

## ****Error Handling in File Operations****

* Always check if fopen() returns NULL to avoid crashes.
* Ensure files are closed properly.

# ****Conclusion****

* **Dynamic memory allocation** allows flexible memory management but requires careful handling to prevent leaks.
* **File handling** enables persistent data storage and retrieval.
* Understanding these concepts helps in writing efficient C programs.

# ****Task:****

****1.** Build a C Program That Still Take Users **Id**, **Name** and **Age** And Store Them in **Array Of Users (User \*ptr)** Until **You Ask To Exit.** After That store Them in **DataBase.txt** Fileand **Print(“Data Stored Successfully”)** if it is stored or **print(“Error Happened During Storing”)** if not.**

****Store Data Format Example:****

****| Id | Name | Age****

****1 Kareem 22****

****2.** Build a C Program That **Read All Data from DataBase.txt and print Them in The Terminal.****

****DeadLine:****

****Next Friday 14/3/2025****

****https://forms.gle/rB2fkRYrHtqavGDx7****