# 1)ENOMEM error in c:

**ENOMEM** is a **macro** defined in C that represents the **error code** for **"Out of Memory"**. It is used by system calls and library functions to indicate that a memory allocation request failed because the system doesn't have enough memory to fulfill the request.

Key Points about `ENOMEM`:

- It is usually defined in the `<errno.h>` header file.
- It has a specific integer value (commonly `12`), but this can vary between systems.
- **Uses-->** It is returned by system calls like `malloc(), calloc()`, or other functions that allocate memory when the system cannot allocate the requested amount of memory.

**A simple program demonstrating ENOMEM:**

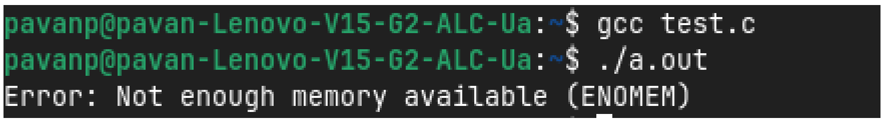
```
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>

int main() {
   // Try to allocate an enormous amount of memory
   int *arr = (int *)malloc(1000000000000 * sizeof(int));
   if (arr == NULL) {
      if (errno == ENOMEM) {
         printf("Error: Not enough memory available (ENOMEM)\n");
      } else {
         printf("Error: Memory allocation failed\n");
      }
   }
   return 0;
}
```

**Check if allocation falied.**

**Allocating a huge amount of memory.**

**Check the error code.**

**O/P**



```
pavanp@pavan-Lenovo-V15-G2-ALC-Ua:~$ gcc test.c
pavanp@pavan-Lenovo-V15-G2-ALC-Ua:~$ ./a.out
Error: Not enough memory available (ENOMEM)
```

## program explaination in points:

- **Memory Allocation Attempt:** The program tries to allocate a very large block of memory.
- **Failure Check**: If the allocation fails, it checks if the failure was due to a lack of memory (`ENOMEM`).
- **Error Handling**: It prints a specific message if there’s not enough memory, otherwise, it prints a general error message.

## 2)Dangling Pointer:

A **dangling pointer** is a **pointer** in C that points to a memory location that has been freed or deallocated. Using a dangling pointer can lead to

- undefined behavior,
- crashes, or
- security vulnerabilities

because it points to **invalid** or **unallocated memory**.

**A simple program on dangling pointer:**

```
#include <stdio.h>
#include <stdlib.h>
int main() {
   int *ptr = (int *)malloc(sizeof(int));            // 1. Allocate memory
   *ptr = 10;                                        // 2. Assign value to allocated memory
   free(ptr);                                        // 3. Free the allocated memory
   ptr = NULL;                                  // 4. Avoid dangling pointer by setting ptr to NULL
   if (ptr == NULL) {                                  // 5. Check if ptr is NULL
      printf("Pointer is NULL and safe.\n");
   }
   return 0;
}
```

## program explaination in points:

- **Memory Allocation**: The program allocates memory dynamically for an integer using `malloc()`, and `ptr` stores the address of this allocated memory.
- **Assigning Value**: The value `10` is stored in the memory location pointed to by `ptr`.
- **Freeing Memory**: The allocated memory is freed using `free(ptr)`, meaning the memory is released and can no longer be used.
- **Avoid Dangling Pointer**: After freeing the memory, `ptr` is set to `NULL` to avoid pointing to invalid (freed) memory.
- **Pointer Check**: The program checks if `ptr` is `NULL`, and if so, it confirms that the pointer is safe to use (no longer a dangling pointer).