

# SO Cheat Sheet

## 4. Gestiunea Memoriei

### WINDOWS

`HANDLE HeapCreate(DWORD flOptions, SIZE_T dwInitialSize, SIZE_T dwMaximumSize)`

- **flOptions** - opțiuni pentru alocarea heapului ( poate fi 0 sau `HEAP_CREATE_ENABLE_EXECUTE`, `HEAP_GENERATE_EXCEPTIONS`, `HEAP_NO_SERIALIZE` )
- **dwInitialSize** - dimensiunea inițială de memorie rezervată heapului (în octeți)
- **dwMaximumSize** - dimensiunea maximă în octeți ( 0 - nu e limitată )
- **întoarce** - handle către noul heap ( NULL în caz de eroare )

`BOOL HeapDestroy(HANDLE hHeap)`

- **hHeap** - handle către heap
- **întoarce** - o valoare diferită de 0 în caz de succes (pentru detalii despre eroare `GetLastError`)

`LPVOID HeapAlloc(HANDLE hHeap, DWORD dwFlags, SIZE_T dwBytes)`

- **hHeap** - handle către heap
- **dwFlags** - suprascrie valoarea specificată de `HeapCreate`: `HEAP_GENERATE_EXCEPTIONS`, `HEAP_NO_SERIALIZE`, `HEAP_ZERO_MEMORY`
- **dwBytes** - numărul de octeți
- **întoarce** - pointer către blocul de memorie

`LPVOID HeapReAlloc(HANDLE hHeap, DWORD dwFlags, LPVOID lpMem, SIZE_T dwBytes)`

- **hHeap** - handle către heap
- **dwFlags** - suprascrie valoarea specificată prin `flOptions`: `HEAP_GENERATE_EXCEPTIONS`, `HEAP_NO_SERIALIZE`, `HEAP_REALLOC_IN_PLACE_ONLY`, `HEAP_ZERO_MEMORY`
- **lpMem** - pointer către blocul de memorie
- **dwBytes** - noua dimensiune a blocului de memorie specificată în octeți
- **întoarce** - pointer către blocul de memorie

În caz de eroare, atât `HeapAlloc`, cât și `HeapReAlloc`, dacă nu s-a specificat `HEAP_GENERATE_EXCEPTIONS`, va întoarce `NULL`, altfel va genera una din următoarele excepții `STATUS_NO_MEMORY` sau `STATUS_ACCESS_VIOLATION`

`BOOL HeapFree(HANDLE hHeap, DWORD dwFlags, LPVOID lpMem)`

- **hHeap** - handle către heap
- **dwFlags** - suprascrie valoarea specificată de `HeapCreate` (`HEAP_NO_SERIALIZE`)
- **lpMem** - pointer către blocul de memorie
- **întoarce** - o valoare diferită de 0 în caz de succes (pentru detalii despre eroare `GetLastError`)

### LINUX

`void *malloc(size_t size)` – alocă memorie neinițializată

`size` numărul de octeți  
`întoarce` un pointer către memoria alocată

`void *calloc(size_t nmemb, size_t size)` – alocă memorie inițializată cu zero

`nmemb` numărul de elemente al vectorului  
`size` dimensiunea în octeți a unui element  
`întoarce` un pointer către memoria alocată

`void *realloc(void *ptr, size_t size)` – modifică dimensiunea blocului de memorie

`ptr` pointer către blocul de memorie  
`size` dimensiunea în octeți  
`întoarce` un pointer către noua zonă de memorie alocată

Atât `malloc`, cât și `calloc` și `realloc` întorc `NULL` în caz de eroare.

`void free(void *ptr)` – dezalocarea unei zone de memorie

`ptr` pointer către zona de memorie

Dacă `ptr` este `NULL` nu se execută nici o operație.

### MTRACE

Trebuie inclusă biblioteca `mcheck.h`

`void mtrace(void)` – activează monitorizarea apelurilor de bibliotecă de lucru cu memoria

`void muntrace(void)` – dezactivează monitorizarea apelurilor de bibliotecă de lucru cu memoria

### GDB

- `gdb "file"`
- `quit`
- `help`
- `run` - pornește execuția
- `kill` - oprește programul
- `break "FUNC"` - setează breakpoint la începutul unei funcții
- `break "ADDR"` - setează breakpoint la adresa specificată
- `nexti "NUM"` - execută NUM instrucțiuni
- `continue` - reia execuția
- `backtrace` - afișează toate apelurile de funcții în curs de execuție
- `info reg` - afișează conținutul registrelor
- `disassemble` - afișează codul mașină generat de compilator

### VALGRIND

`valgrind --tool=memcheck ./executabil`