

C Programming Essentials: Macros, Memory, and More

Quick overview of key C concepts including macros, dynamic memory, preprocessor directives, and header guards.



by Hamdi Emad

```
Fest
Custrast Organicclesar()
Clnce
Clice: Instalsticiss(pocke = faglesty 1;
restins. & le:
arstactialt);
inglarte: Clasce Tadme = Meulatlle-BictaclDS/);
preteres: Cactalablenltather)

tentDriak audl reeffactvel = VP descrtatless(Uidldillag))

(= Flasiuder: Compiter(lateditiy/);
fonttaolk audl rate: Satal ty Rade = {Of Inststand)

tencDriok andl reeffactvel = VP Excibletligr))
| Rig;
| defilgater-1, OSr Instosthe: (
| etidlense: reednbableycches: ( Gatile:/Be(Pugio((IngUaeT)), (:=)
| crctcion(7alecrlies) , <=;

(> Stocks Interter: Mewest =30:
instplejone ()
```

Imest Wotks

C Macros: Definitions and Usage

What is a Macro?

A preprocessor directive for code substitution before compilation.

How to Define

Use **#define**, e.g., *#define PI 3.14159* or *#define SQUARE(x) ((x)*(x))*.

Purpose

Simple text replacement increases code flexibility and reuse.

```
1  #include <stdio.h>
2
3  #define PI 3.14159265
4
5  int main() {
6
7  }
8
9
```

Macros vs. Functions: Key Differences

Macros

- Preprocessor substitution, inline expansion
- No type checking
- Faster but may cause code bloat

Functions

- Compiled code with type safety
- Function call overhead
- Better debugging and safety

Conditional Compilation: #ifdef, #ifndef, #endif

#ifdef

Checks if a macro is defined;
enables conditional code.

#ifndef

Checks if a macro is NOT
defined; used for header
guards.

#endif

Ends the conditional compilation block.

Dynamic Memory Allocation: malloc()

malloc()

Allocates memory at runtime.

Returns

A **void*** pointer; requires explicit casting.

Usage

Use **sizeof** to specify bytes needed.

malloc(), calloc(), realloc(): Comparison

`malloc()`

Allocates uninitialized memory.

`calloc()`

Allocates zero-initialized memory.

`realloc()`

Resizes previously allocated memory block.

The Importance of free()

Manual Memory Release


Dynamic memory must be freed to avoid leaks.

free()

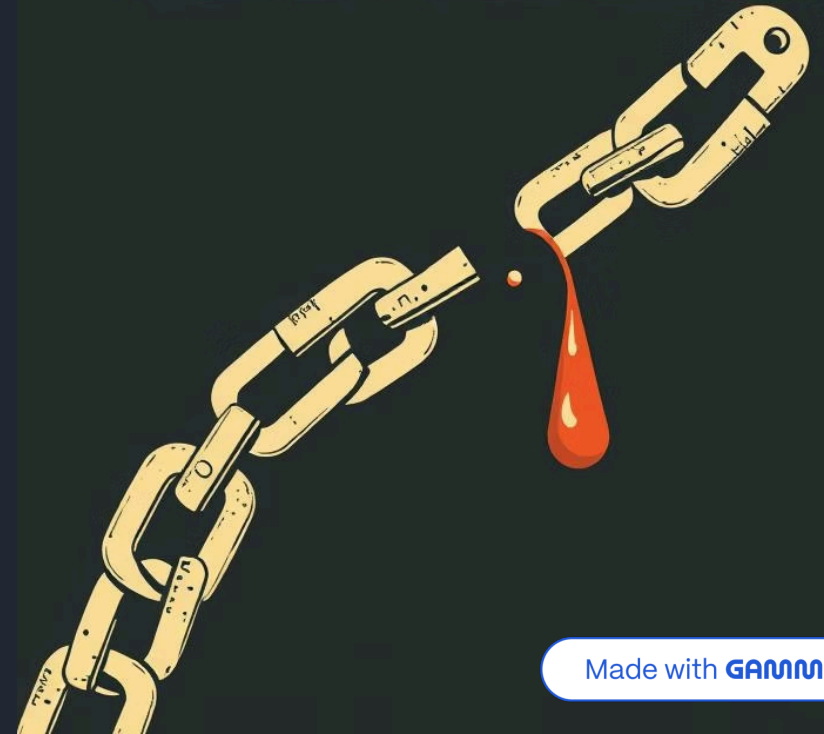
Deallocates memory allocated by malloc/calloc/realloc.

Consequences

Failing to free causes memory leaks and resource waste.



MEMORY LEAK




```

17  #iedfe      #fftccoIocera(tapor:seeclcop:detchi:cca(ocal):
10  #iendif    #ffinc.Zeacelotrecheist;
18  #iendif    #fftcebIoccert(resce:stta/cratalelis::

19  #iindf      #fftcebLog:roprlacce:strylindesith tytichlyccelat:
17  #iledif    #fftconIourind.:itene:sttalstgearice(com_ saele:
28  #itndf     #fftcerTeocert(leenloattle,Ihatrlotrett/cto/ldl/syebolitanys.gedif)
33  #iinff     #fftccI Fexcelo(rspor:staclotritserlecsigoof/ant:ccfiet:
34  #iindif    #fftcenFou:ing. Ideabina(ocd.elle.lot);
35  #ilooif    #ffceC. TeacelablT,colastalccalrylicbtal gooffclofil/-cltgltter):
30  #iinff     #ffrcchlegccrrfrecnaist:

```

Header Guards: Preventing Multiple Inclusions

What Are Header Guards?

Conditional compilation to avoid double header inclusion.

Benefits

Prevents redefinition and compilation errors.

Usage

Essential for stable large-scale project compilation.

Header Guard Format: The Standard Approach

```
#ifndef UNIQUE_HEADER_NAME_H
#define UNIQUE_HEADER_NAME_H
    ...header content...
#endif
```

The unique name prevents conflicts by ensuring one-time inclusion.

```
#ifndef MATH_UTILS_H
#define MATH_UTILS_H

int add(int a, int b);
int subtract(int a, int b);

#endif // MATH_UTILS_H
```

Nested Includes: Preprocessor Handling

Include Recursion

Preprocessor recursively processes included files.

Role of Header Guards

Prevents infinite inclusion loops.

Compilation Order

Include order affects symbol visibility and compilation.