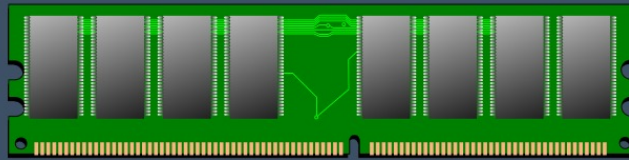


[Home](#)[Kelas](#)[Tutorial](#)[Buku](#)[Discord](#)

Ahmad Muhardian · 15 May 2022

Belajar C #16: Fungsi untuk Alokasi Memori Secara Dinamis

#C



```
malloc()    calloc()
realloc()   free()
```

petanikode.com

Pada tutorial ini kita akan belajar gimana cara mengalokasikan memori untuk data dinamis dengan fungsi `malloc()`, `calloc()`, `realloc()`, dan `free()`.

Mengapa kita membutuhkan fungsi ini?

dan gimana cara pakainya?

Mari kita pelajari!

Ad

Mengapa Kita Membutuhkan Malloc()?

Saat kita menjalankan program, komputer akan mengalokasikan bagian:

1. Bagaian untuk menyimpan kode (Stack Code);

[> Baca selengkapnya](#)

Hindari kontak dengan orang yang sakit, termasuk jangan saling berjabat tangan, berciuman atau berpelukan.

2. Bagian untuk menyimpan variabel global atau static seperti konstanta;
3. Bagian untuk menyimpan variabel lokal, area ini bisa kita sebut stack;
4. Bagian untuk menyimpan variabel dengan alokasi dinamis (*heap*).

Perhatikan gambar ini:



Bagian **Code** akan menyimpan kode instruksi dari program. Kemudian bagian **Global**, **Stack**, dan **Heap** akan menyimpan nilai dari variabel.

Kira-kira isinya mungkin akan seperti ini:



Saat kita membuat program seperti ini:



```
#include <stdio.h>

int score = 0;

void main() {
    int level = 1;
}
```

Maka semua baris instruksi akan disimpan ke dalam **Stack Code**. Nilai dari variabel **score** akan disimpan pada bagian stack **Static/Global** dan nilai dari variabel **level** akan disimpan di dalam **Stack**.

Isi Stack Code:

Address	Content
0000	#include <stdio.h>
0001	int score = 0
0002	void main();
0003	int level = 1

Isi Stack Global:

Address	Content
0000	null
000a	0

Anggap saja **000a** adalah alamat memori dari variabel **score**.

Isi Stack Lokal:

Address	Content
0000	null
000b	1

Anggap saja **000b** adalah alamat memori dari variabel **level**.

Sementara area **Heap** dipakai untuk menyimpan data yang ukurannya dinamis dan akan disimpan di lokasi yang acak.

Contoh data dinamis kayak gimana?

Data yang ukurannya tidak tetap.

Contoh:

```
// ini variabel dengan ukuran tetap
int enemies[10];

// ini ukuran variabel dengan ukuran dinamis
char name[];
```

Variabel **enemies** akan berukuran $10 * 4$ byte, yakni **400** byte. Ini karena tipe data integer ukuran default-nya adalah **4** byte. Saat kita membuat array dengan isi 10 integer, maka ukurannya akan 10 kali lipat.

Artinya, variabel **enemies** sudah kita alokasikan ukurannya $10 * 4$ byte dan ini tidak bisa berubah secara dinamis.

Misalnya, kita ingin isi variabel **enemies** dengan **15** item, maka ini tidak akan bisa.. karena ukurannya sudah dibatasi **10**.

Sementara untuk variabel **name**, bisa kita isi dengan panjang berapapun karena ukurannya dinamis.

Tapi..

Membuat variabel seperti ini:

```
char name[];
```

Akan membuatnya disimpan ke dalam Stack, bukan Heap.

Memangnya kenapa kalau disimpan di dalam Stack?

Stack punya batasan yang sudah ditentukan oleh sistem operasi.

Misalnya, anggap saja ukuran stack kita **100** MB. Lalu kita mengisi variabel **name** dengan teks yang ukurannya **500** MB.

Maka apa yang akan terjadi?

“Stack Overflow”

Yap, nama StackOverflow diambil dari istilah ini.

Eh, kok malah bahas StackOverflow. 😊

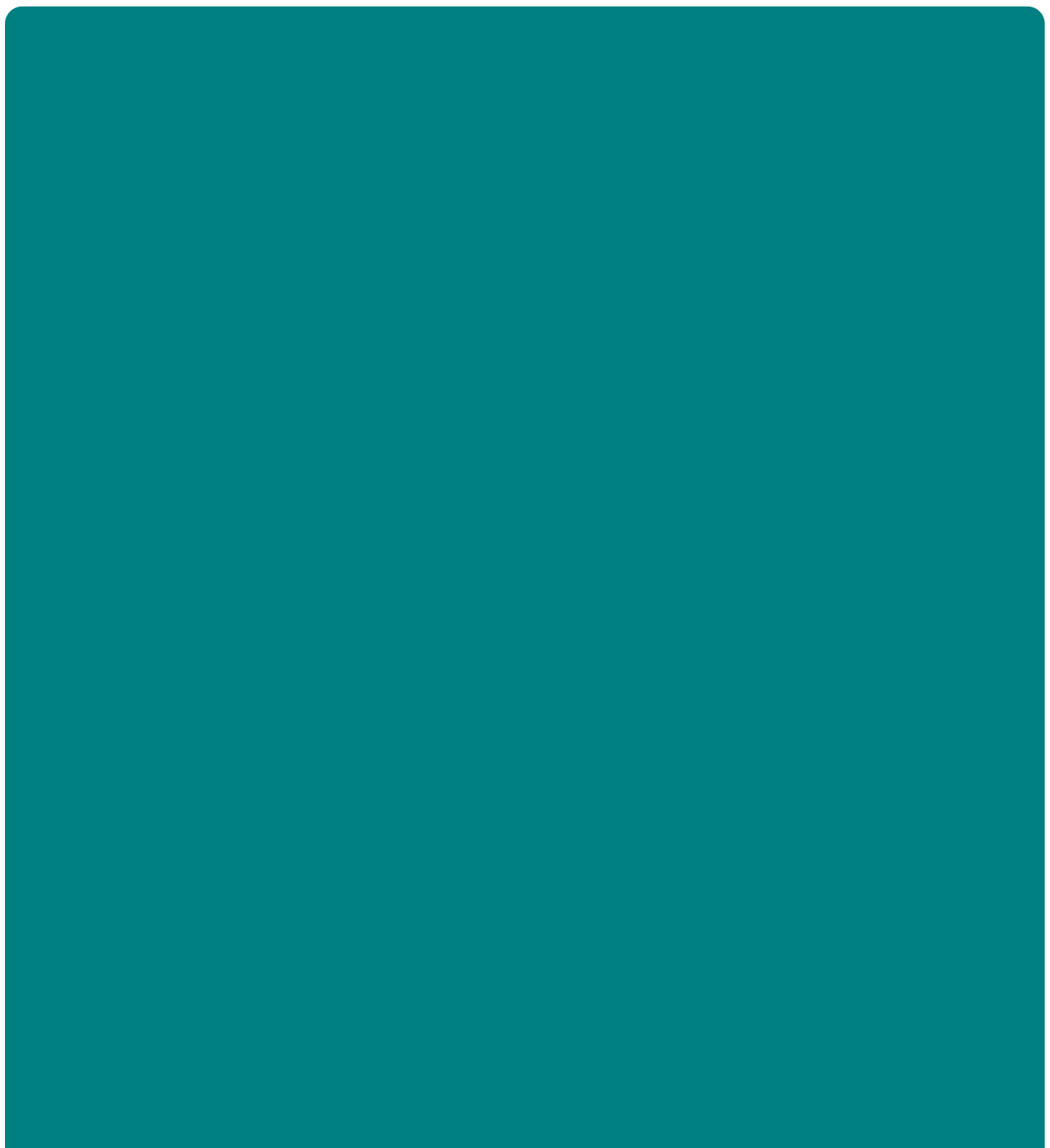
Masalah Stack Overflow adalah masalah saat kita memberikan data yang melebihi ukuran stack pada memori.

Biar tidak terjadi Stack Overflow, maka kita harus pakai memori Heap.

Gimana Caranya?

Caranya menggunakan fungsi untuk alokasi memori seperti `malloc()`, `calloc()`, `realloc()`, dan `free()`.

Mari kita pelajari..



Mengenali Fungsi `malloc()`

Fungsi `malloc()` merupakan fungsi untuk mengalokasikan memori secara dinamis dan datanya akan disimpan pada memori **heap**.

Fungsi ini berada di dalam library `stdlib.h`, jadi jika ingin pakai fungsi `malloc()` maka library `stdlib.h` harus kita import terlebih dahulu dengan `#include`.

Contoh:

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

Barulah setelah itu kita bisa pakai fungsi `malloc()`.

Cara menggunakannya gimana?

Fungsi `malloc()` menghasilkan sebuah pointer. Pointer tersebut berisi alamat memori pada heap.

Berikut ini format penggunaan `malloc()`:

```
type *nama_var = malloc(ukuran);
```

Penjelasan:

- `type` adalah tipe data variabel yang ingin kita buat;
- `*nama_var` adalah variabel yang bentuknya pointer;
- `ukuran` adalah ukuran alokasi memori dalam satuan byte dengan tipe integer.

Contoh:

```
int *score = malloc(32);
```

Pada contoh ini, kita membuat variabel `*score` dengan tipe data `int` dan akan disimpan ke dalam heap dengan ukuran `32` byte.

Jika ingin ukurannya mengikuti ukuran tipe data, kita bisa gunakan fungsi `sizeof()` seperti ini:

```
int *score = malloc(sizeof(int));
```

Artinya ukuran variabel `*score` di Heap akan mengikuti ukuran tipe data `int`, yakni `4` byte.

Fungsi `sizeof()` biasanya kita pakai untuk menentukan ukuran secara dinamis.

Misalnya kita meminta user untuk menginputkan namanya dan kita bisa mengalokasikan ukuran memori berdasarkan panjang namanya dengan fungsi `sizeof(name)`.

Contoh:

```
char name[] = "Petani Kode"; // diinputkan user

// alokasi memori dinamis
char *name = malloc(sizeof(name));
```

Oke..

Biar makin paham, mari kita latihan!

Latihan: Fungsi `malloc()`

Buatlah program baru dengan nama `contoh_malloc.c` kemudian isi dengan kode berikut:

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

void main(){
    struct Player {
        char *name;
        unsigned int hp;
        unsigned int xp;
        unsigned int level;
    };

    // menggunakan malloc
    struct Player *player1 = malloc(sizeof(struct Player));

    player1->name = "Petani Kode";
    player1->hp = 100;
    player1->xp = 5;
    player1->level = 1;

    printf(":: PLAYER STATUS ::\n");
    printf("name : %s\n", player1->name);
    printf("hp   : %d\n", player1->hp);
    printf("xp   : %d\n", player1->xp);
```

```
printf("level: %d\n", player1->level);  
}
```

Setelah itu, compile dan jalankan.

Maka hasilnya:



Pada contoh ini, kita menggunakan fungsi `malloc()` untuk mengalokasikan memori pada variabel `*player`. Ukuran alokasi memorinya akan mengikuti ukuran dari struct `Player`, karena kita menggunakan fungsi `sizeof()` di sana.

Mengenal Fungsi `calloc()`

Fungsi `calloc()` sama seperti fungsi `malloc()`, sama-sama berfungsi untuk mengalokasikan memori pada Heap.

Bedanya, `calloc()` menggunakan beberapa blok memori untuk satu variabel, sedangkan `malloc()` hanya mengalokasikan satu blok dengan ukuran tertentu untuk satu variabel.

Coba perhatikan gambar ini:



Nilai **100** dan **123** adalah satu blok yang ukurannya sudah ditentukan oleh **malloc()**. Sedangkan teks **"Petani Kode"** adalah dua blok memori yang dibuat dengan **calloc()**.

Performa fungsi **calloc()** lebih lambat dibandingkan **malloc()**, karena ia menggunakan banyak blok memori.

Cara menggunakan fungsi **calloc()**, sama seperti **malloc()** hanya saja beda parameter yang diberikan.

Fungsi **calloc()** punya dua parameter:

```
calloc(jumlah_blok, ukuran);
```

Penjelasan:

- **jumlah_blok** adalah jumlah blok yang ingin dibuat (integer);
- **ukuran** adalah ukuran tiap blok dalam satuan byte (integer).

Fungsi **calloc()** juga menghasilkan sebuah pointer, karena itu kita harus menyimpannya dalam variabel pointer.

Contoh:

```
char *name = calloc(2, 32);
```

Oke, biar semakin paham..

Mari kita latihan!

Buatlah program baru dengan nama **contoh_calloc.c**, kemudian isi dengan kode berikut:

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

void main(){
    struct Product {
```

```

    char *name;
    unsigned int price;
    unsigned int stock;
    float weight;
};

// menggunakan calloc
struct Product *buku = calloc(2, sizeof(struct Product));

buku->name = "Pemrograman C untuk Pemula";
buku->price = 98000;
buku->stock = 5;
buku->weight = 1.2;

printf("## DETAIL PRODUK ##\n");
printf("name : %s\n", buku->name);
printf("harga: %d\n", buku->price);
printf("stok : %d\n", buku->stock);
printf("berat: %.2f kg\n", buku->weight);
}

```

Setelah itu, compile dan jalankan.

Maka hasilnya:



Pada contoh ini, kita menggunakan fungsi `calloc()` untuk mengalokasikan memori variabel `buku` sebanyak `2` blok dengan ukuran mengikuti ukuran struct `Product`.

Mengenal Fungsi `realloc()`

Fungsi `realloc()` adalah fungsi untuk mengalokasikan ulang memori dari variabel yang sudah dialokasikan dengan fungsi `malloc()` dan `calloc()`.

Ini biasanya kita butuhkan saat kita ingin menambah atau mengurangi ukuran alokasi memori pada suatu variabel.

Fungsi `realloc()` memiliki dua parameter yang harus diberikan:

```
realloc(*pointer, ukuran_baru);
```

Penjelasan:

- `*pointer` adalah variabel pointer yang ingin kita alokasikan ulang;
- `ukuran_baru` adalah ukuran alokasi memori untuk mengalokasikan ulang variabel tersebut.

Contoh:

```
// mengalokasikan 16 byte
char *name = malloc(sizeof(char) * 16);

// mengalokasikan ulang menjadi 32 byte
name = realloc(name, sizeof(char) * 32);
```

Ukuran 1 karakter untuk tipe data `char` adalah 1 byte. Pada contoh ini kita mengalokasikan memori untuk `16` karakter atau 16 byte. Lalu mengalokasikan ulang dengan `realloc()` menjadi `32` karakter.

Oya, fungsi `realloc()` juga akan mengubah alamat memori dari variabel ke alamat yang baru.

Biar semakin paham, mari kita coba dalam program.

Buatlah program baru dengan nama `contoh_realloc.c`, kemudian isi dengan kode berikut:

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

void main(){
    /* Initial memory allocation */
    char *str = malloc(15);
    strcpy(str, "petanikode");
    printf("String = %s, Address = %x\n", str, str);

    /* Reallocating memory */
    str = realloc(str, 25);
```

```
    strcat(str, ".com");  
    printf("String = %s, Address = %x\n", str, str);  
}
```

Setelah itu, compile dan jalankan.

Maka hasilnya:



Mengenal Fungsi `free()`

Fungsi `free()` adalah fungsi untuk menghapus alokasi memori yang sudah dibuat oleh fungsi `malloc()`, `calloc()`, dan `realloc()`.

Fungsi `free()`, punya satu parameter yakni variabel yang ingin dihapus alokasinya.

```
free(nama_variabel);
```

Biar lebih jelas, mari kita coba!

Ubahlah program pada latihan sebelumnya (`contoh_realloc.c`) menjadi seperti ini:

```
#include <stdio.h>  
#include <stdlib.h>  
  
void main(){  
    /* Initial memory allocation */  
    char *str = malloc(15);  
    strcpy(str, "petanikode");
```

```
printf("String = %s, Address = %x\n", str, str);

/* Reallocating memory */
str = realloc(str, 25);
strcat(str, ".com");
printf("String = %s, Address = %x\n", str, str);

free(str);

printf("String = %s, Address = %x\n", str, str);
}
```

Setelah itu, compile dan jalankan.

Maka hasilnya:



Isi variabel `str` akan kosong, karena alokasi memorinya sudah kita hapus. Namun, dia tetap melakukan pointing pada alamat memori di Heap.

Apa Selanjutnya?

Sejauh ini kita sudah belajar melakukan alokasi memori dengan fungsi `malloc()`, `calloc()`, `realloc()`, dan `free()`.

Selanjutnya silakan latihan dan terapkan fungsi ini di program yang kamu buat.

Jika masih bingung, coba tanyakan di komentar.

Selamat belajar. 🙏

Daftar isi tutorial


 Belajar Pemrograman C #01: Pengenalan Bahasa Pemrograman C

 Belajar Pemrograman C #02: Persiapan Pemrograman C di Linux

 Belajar Pemrograman C #02: Persiapan Pemrograman C di Windows


 Belajar Pemrograman C #03: Struktur Dasar Penulisan Program C

 Belajar Pemrograman C #04: Fungsi Input dan Output


 Belajar Pemrograman C #05: Variabel, Konstanta, dan Tipe Data


 Belajar Pemrograman C #06: Operator


 Belajar Pemrograman C #07: Percabangan


 Belajar Pemrograman C #08: Perulangan


 Belajar Pemrograman C #19: Struktur Data Array

 Belajar Pemrograman C #10: Prosedur dan Fungsi


 Belajar Pemrograman C #11: Tipe Data Enum

 Belajar Pemrograman C #12: Tipe Data Structure

 Belajar Pemrograman C #13: Tipe Data Union

 Belajar Pemrograman C #14: Tipe Data String

 Belajar Pemrograman C #15: Apa itu Pointer?

 Belajar Pemrograman C #16: Fungsi untuk Alokasi Memori

 Belajar Pemrograman C #17: Cara Membaca dan Menulis File di C

 Belajar Pemrograman C #18: Memahami Preprocessor dan Macro



Newsletter..

Mau dapat tips belajar coding, info teknologi, dan perkembangan karir sebagai programmer?

 [Subscribe](#)



Artikel Terbaru

Belajar C++ #13: Mengenal Tipe Data Union

03 May 2024 • baca 5 menit



Belajar C++ #12: Mengenal Tipe Data Struct

01 May 2024 • baca 8 menit



Belajar C++ #11: Tipe Data Enum di C++

28 Apr 2024 • baca 6 menit



Belajar C++ #14: Memahami Pointer di C++

23 Nov 2023 • baca 10 menit



Cara Install dan Setup Unity Engine di Mac dengan Benar

23 Nov 2023 • baca 5 menit



Tutorial CSS: Menentukan Ukuran Elemen dengan Satuan yang Tepat

28 Mar 2023 • baca 8 menit



Powered by **GliaStudio**



Tempat belajar budidaya kode (coding)
dengan tutorial yang gampang dipahami.

Belajar

Artikel

Popular Tutorial

Tutorial Bahasa C

Social Media

Facebook Page

Petani Kode

About

[Tutorial](#)

[Tutorial Javascript](#)

[Instagram](#)

[FAQs](#)

[Buku](#)

[Tutorial Java](#)

[Twitter](#)

[Contact](#)

[Tutorial PHP](#)


[Youtube Channel](#)

[Tutorial Python](#)

[Telegram Channel](#)

Ikuti Kami di



© 2024 **Petani Kode** · Made with  using Hugo 0.123.8