



CPP Module 00

Yapıcı Fonksiyon (Constructor)

Bir nesneyi tanımlarken PhoneBook phoneBook dersek mesela phoneBook classın objesi ya

PhoneBook classını buluyor ve oradaki constructor çalışıyor. Yapıcı fonksiyon

Constructor fonksiyonu, nesne oluşturulduğunda ilk çalışacak fonksiyondur. Class ile aynı isimde olmak zorundadır. Constructor herhangi bir tipte değildir.

```
constructor > G+ main.cpp > ...
1  #include <iostream>
2  #include <string>
3  #include <iomanip>
4
5  class PhoneBook
6  {
7  public:
8
9      PhoneBook()
10     {
11         std::cout << "Constructor cagirildi" << std::endl;
12     }
13 };
14
15 int main()
16 {
17     PhoneBook phoneBook; //Nesneyi tanımlarken çalışan ilk fonksiyondur. Sınıfla aynı isimdirConstructor Kurucu
18     PhoneBook phoneBook1;
19
20     return(0);
21 }
22
```

SORUNLAR ÇIKIŞ HATA AYIKLAMA KONSOLU **TERMINAL** MEMORY XRTOS

```
• → constructor ./a.out
Constructor cagirildi
Constructor cagirildi
○ → constructor
```

Yıkıcı Fonksiyonlar (Destructor)

Bir nesne yok edilmeden hemen önce yıkıcı fonksiyon çalışır. Buna örnek vermek gerekirse. Php'de genelde işlemler şöyle yapılır. Constructor içerisinde mysql connect işlemleri yapılır. Destructor içerisinde ise mysql'i kapatma işlemleri yapılır.

Yıkıcı fonksiyon yazarken ~ bu işlem ile başlıyor.

std::cin >> str → Bunda girilen değerler boşlukluysa patlıyor.

Bunun muadili std::getline(std::cin,str); Bu kullanılır boşlukları da okuyor mekanizması get_nextlinedaki buffer mekanizmasına dayanıyor.

Stack & Heap (Yığın)

Yazılım geliştiriyor veya bu alanda eğitim görüyorsanız muhakkak stack ve heap kavramları ile karşılaşmışsınızdır. Peki nedir bu stack ve heap?

Kodumuz işletim sisteminde bir yer kaplar. Bu yerin boyutu kimi zaman belirli yani değişmez iken kimi zaman ise kullanıcının program esnasında gireceği verilere göre değişebilecek durumdadır. Temel olarak bu farklılıktan dolayı iki farklı yöntem mevcut.

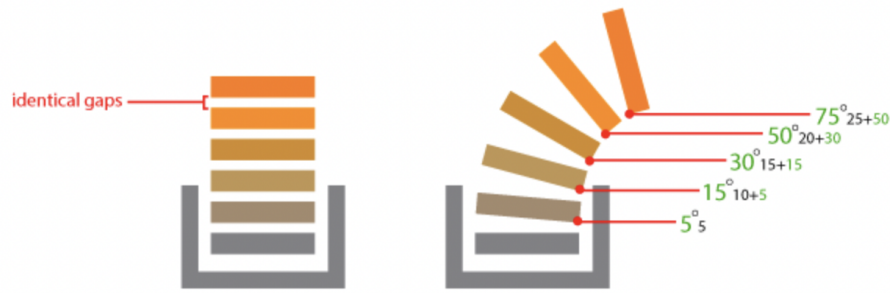
Bir diğer deyişle değerlerin RAM de saklandığı 2 kısım vardır: Stack ve Heap. İki kelime de Türkçeye 'yığın' olarak çevrilse de aralarında çok önemli farklar bulunuyor. Kısaca Stack için, boyutları belli sabit değerlerin saklandığı kısım ve Heap için de, değişken değerlerin saklandığı kısım demek mümkün.

Heap ve Stack arasında ki en önemli farklardan birisi heapde veriler karışık bir şekilde saklanırken stackte artan ya da azalan adres mantığında (big and little endian) çalışır. Buna bağlı olarak heapde yer alan bir veriye erişmek stackte yer alan bir veriye erişmeye göre daha maliyetli bir işlemdir. Başka bir fark ise stackteki veri hemen silinirken heapdeki veri Garbage Collector algoritmasına bağlıdır.

Stack bellekten statik olarak yer tahsisi için kullanılırken, Heap dinamik olarak yer tahsisi etmeyi sağlar. Her ikisi de Ram bölgesinde bulunur. Stack'te yer alan veriler direk bellek içine yerleştirilir dolayısıyla erişimi çok hızlıdır. Heap ise runtime (çalışma zamanı) anında kullanılırlar ve dağınık bir bellek göz yapısı olduğu için erişimi stack kadar kolay olmaz dolayısıyla yavaş çalışır. Stack bellekteki veri hemen silinirken Heap bellekteki verinin silinmesi Garbage Collector'a (Çöp toplama mekanizmasına) bağlıdır. Stack alanı sınırlı olduğundan çok büyük sayıda ve büyük tiplerde veri atanması belleğin dolmasına sebep olabilir.

Stack veri yapısına üst üste dizili tabaklar örnek verilebilir. Alttaki bir tabağı almak istediğinizde nasıl ki üstündeki tabakları da indirmeniz gerekiyorsa, stack veri yapısında da aradaki bir veriyi alabilmek için öncelikle üsttekileri çekmek gerekiyor.

Kullanacağınız yerin boyutunu tam olarak biliyorsanız *Stack*, ihtiyacınız olan boyutu tam olarak bilmiyorsanız *Heap* kullanımı daha mantıklı bir tercih olacaktır.



Stack'ın Heap'den bir farkı da önceden belirlenmiş bir boyutu olmasıdır, yani belirli miktarda veriyi tutabilir. Bu boyut windows'ta default olarak 1mb'dır, fakat bazı Unix sistemlerinde bu boyut 8 mb'a kadar çıkabilir. Eğer program Stack'e bu boyutun kaldırabileceğinden fazla veri koymaya çalışırsa **Stackoverflow** durumuyla karşılaşılır.

Verimli **kodlama** yapabilmek için yazdığımız kodlar çalıştığı esnada arka planda neler olduğunu bilmemiz gerekiyor. Yazdığımız kodların hepsi RAM da yer kaplamakta. Performansın önemi ufak programlarda çok hissedilmese de profesyonel çalışmalarda hayati bir etkiye sahip.

What does Delete Mean ?

- Delete can be used by either using **Delete operator** or **Delete operator**
- New operator is used for dynamic memory allocation which puts variables on heap memory.
- Which means Delete operator deallocates memory from heap.
- Pointer to object is not destroyed, value or memory block pointed by pointer is destroyed.
- The delete operator has **void** return type does not return a value.
-


```
// Program to illustrate deletion of array
#include <bits/stdc++.h>
using namespace std;




int main()
{
    // Allocate Heap memory
    int* array = new int[10];
    // Deallocate Heap memory
```

```
delete[] array;

return 0;
}
```

```
#include <bits/stdc++.h>
using namespace std;
```



```
int main()
{
    // Creating int pointer
    int* ptr1 = new int;

    // Initializing pointer with value 20
    int* ptr2 = new int(20);

    cout << "Value of ptr1 = " << *ptr1 << "\n";
    cout << "Value of ptr2 = " << *ptr2 << "\n";

    delete ptr1; // Destroying ptr1
    delete ptr2; // Destroying ptr2

    return 0;
}
```

Output:

```
Value of ptr1 = 0
Value of ptr2 = 20
```

4. Deleting a *void* pointer

CPP

```
#include <bits/stdc++.h>
using namespace std;
int main()
{
    void* ptr; // Creating void pointer

    delete ptr; // Destroying void pointer

    cout << "ptr deleted successfully";
    return 0;
}
```

Output:

```
ptr deleted successfully
```

exceptions.

1. Trying to delete Non-pointer object

CPP

```
#include <bits/stdc++.h>
using namespace std;
int main()
{
    int x;

    // Delete operator always
    // requires pointer as input
    delete x;

    return 0;
}
```

Output:

```
error: type 'int' argument given to 'delete', expected pointer
```

C++ is an object-oriented programming language.

Everything in C++ is associated with classes, objects, attributes, and methods. For example: in real life, a car is an **object**. The car has **attributes**, such as weight and color, and **methods**, such as drive and brake.

Attributes and methods are basically **variables** and **functions** that belong to the class. These are often referred to as "class members".

A class is a user-defined data type that we can use in our program, and it works as an object constructor or a "blueprint" for creating objects.

std::cin'den sonra kullanılan getline cin'den gelen '\n' karakterini algılıyor, bu yüzden beklemeden boş string tutuyor. BİR ARADA KULLANMA! (açıklama aşağıda)



Note that just inputting `std::cin` directly into an integer does not work; it will leave the trailing '\n' character in the buffer, which will desynchronize the following input. It's possible to avoid this by adding a call to `std::cin.ignore`, but this would still miss the error due to trailing garbage. If you're doing line-oriented input, stick with `getline`, and use `std::istream` for any necessary conversions.