ALGORİTMALAR VE PROGRAMLAMA II

Hafta 2
C Programlarının Bellek Düzeni ve Rekürsif (Özyinelemeli) Fonksiyonlar

hru-algpro.github.io

Saklama Sınıfları

- Dört depolama sınıfı **auto**, **extern**, **register** ve **static** olarak verilir.
  - **auto:** Fonksiyon içinde bildirilen değişkenler varsayılan olarak otomatiktir. Bu değişkenler fonksiyon kapsamında kullanılabilir. auto double x, y;

    -Stack bölgesinde tutulurlar.
    -Global değişkenler ve parametre değişkenleri auto özelliği alamaz.
  - **extern:** Bloklar ve fonksiyonlar arasında bilgi aktarma yöntemlerinden biri harici değişkenleri kullanmaktır.

    -**extern** ile tanımlanan değişkene başlangıç değeri verilmez ise derleyici tarafından hafızada yer ayrılmaz.

Örnekler: auto & extern

- Extern'in bu kullanımı derleyiciye bu dosyada veya başka bir dosyada "başka yerde arama" yapmasını söylemek için kullanılır.

ornek_kod.c

```c
#include <stdio.h>

int a = 1, b = 2, c = 3;
int getInt(void);

int main(void) {
    printf("%3d\n", getInt());
    printf("%3d%3d%3d\n", a, b, c);
    return 0;
}
```

ornek_kod_2.c

```c
int getInt(void) {
    extern int a;  // Must specify 'int'
    int b, c;
    b = c = a;
    return (a + b + c);
}
```

Saklama Sınıfları

- **register:** Saklama sınıfı register, derleyiciye ilişkilendirme değişkenlerinin yüksek hızlı bellek kayıtlarında saklanması gerektiğini söyler.

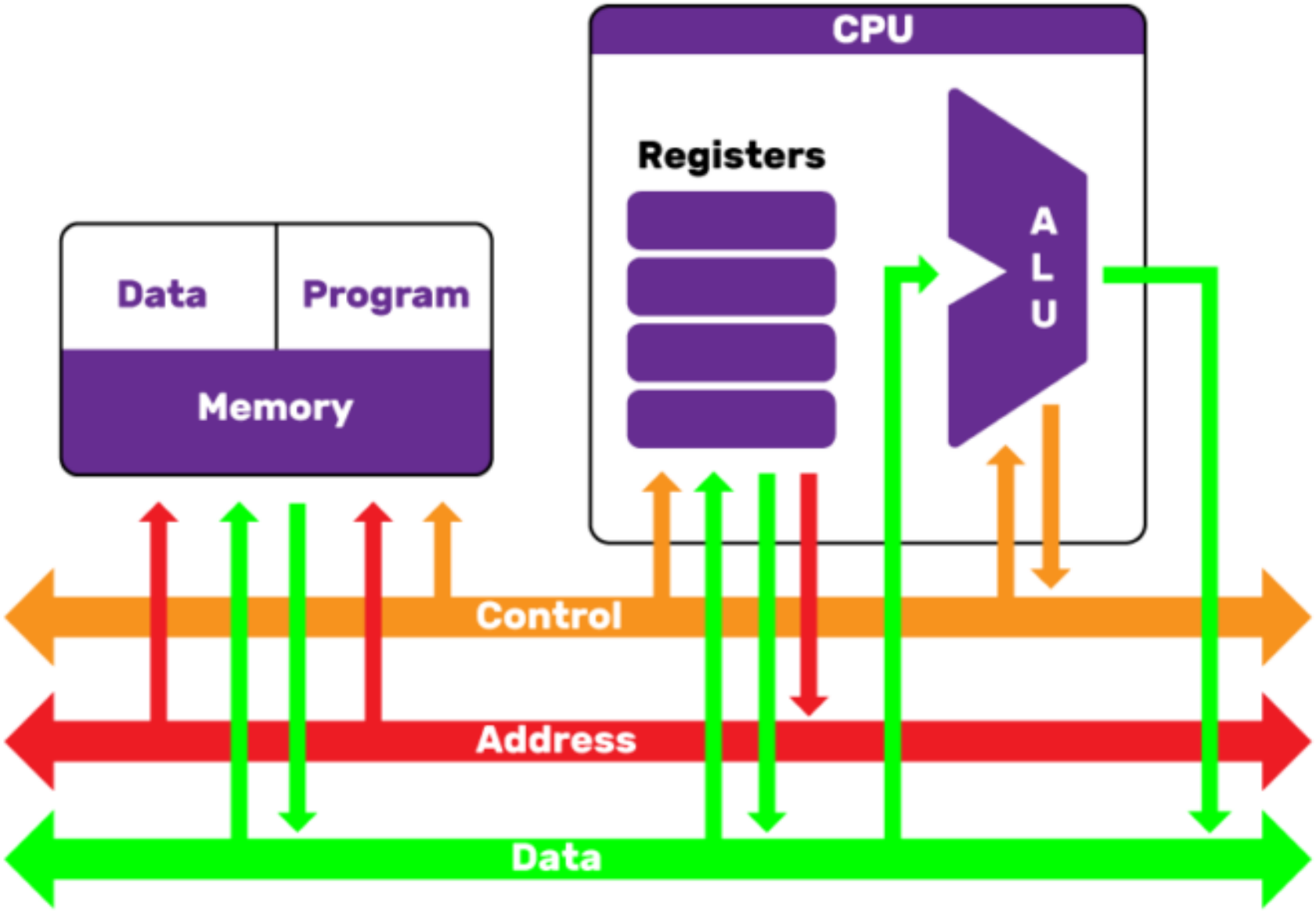
  Belleğe erişim, yazmaçlara erişimden daha yavaştır. Çünkü belleklere erişim için belli bir makine zamanı gerekir.

- **static:** Fonksiyonlar içinde tanımlanan yerel değişkenlerdir.

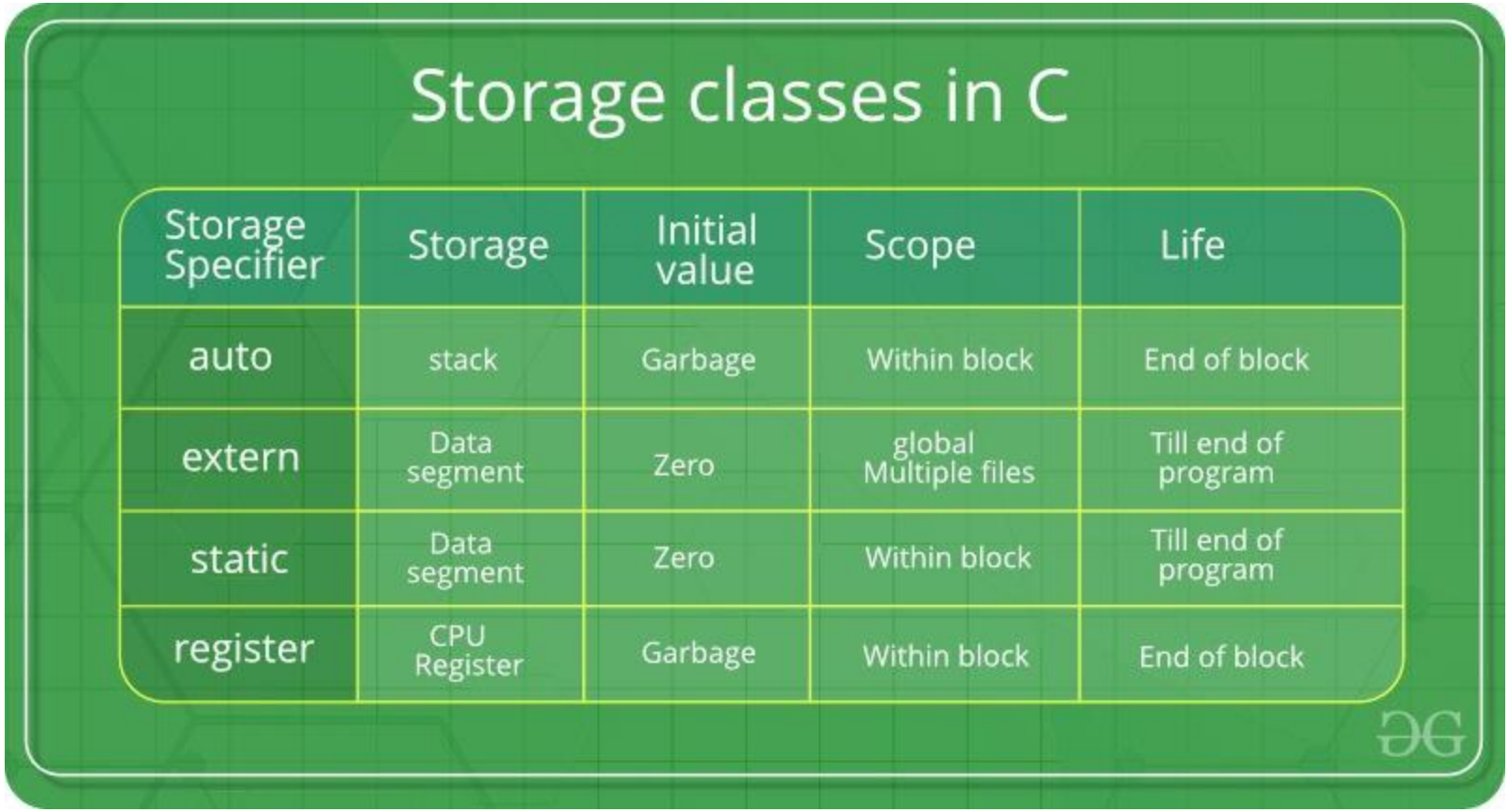
  Fonksiyon sonlandıktan sonra değişken değeri saklanır.

  Sadece tanımlandıkları fonksiyonda geçerlidirler.

  Statik yerel ve global değişkenler **data segment** bölgesinde tutulur.


CPU
Registers
ALU
Data
Program
Memory
Control
Address
Data

Saklama Sınıfları

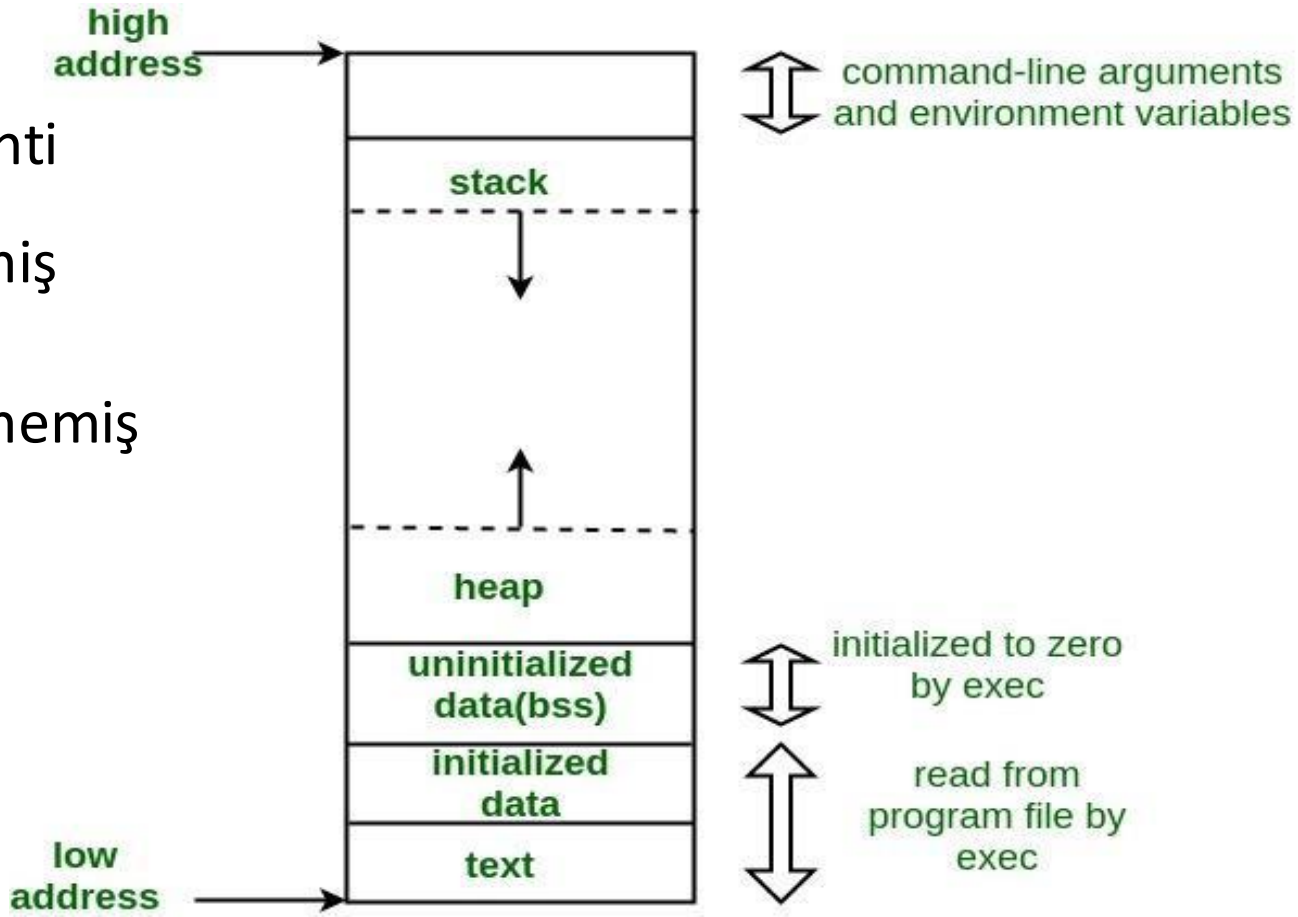


https://www.geeksforgeeks.org/storage-classes-in-c/

C Programlarının Bellek Düzeni

- C programının tipik bir hafıza temsili aşağıdaki bölümlerden oluşur.

1. Metin segmenti
2. İlk değer verilmiş veri segmenti
3. İlk değer verilmemiş veri segmenti
4. Yığın (Stack)
5. Öbek (Heap)



C Programlarının Bellek Düzeni

1. Metin (Kod) Segmenti:

- Bir programın bir nesne dosyasındaki veya bellekteki yürütülebilir komutları içeren bölümlerinden biridir.

- Derlenen programın makine kodunu(Assembly) tutar.

- Genellikle, metin kesimi paylaşılabilir olduğundan, metin düzenleyicileri, C derleyicisi, kabukları vb. gibi sık kullanılan programlar için tek bir kopya bu paylaşımlı bölgede bulunur.

- Ayrıca, bir programın talimatlarını yanlışlıkla değiştirmesini önlemek için metin segmenti salt okunurdur.

C Programlarının Bellek Düzeni

2. İlk Değer Verilmiş Veri Segmenti:

- Veri segmenti, programcı tarafından başlatılan global değişkenleri ve statik değişkenleri içeren bir programın sanal adres alanının bir kısmıdır.

3. İlk Değer Verilmemiş Veri Segmenti:

- Bu bölümdeki veriler, program başlatılmamış verileri çalıştırmaya başlamadan önce çekirdek tarafından aritmetik ‘0’ (sıfır) olarak başlatılır. Veri bölümünün sonunda başlar ve tüm global değişkenleri ve sıfır olarak başlatılmış veya kaynak kodda açık bir şekilde başlatılmamış statik değişkenleri içerir.

C Programlarının Bellek Düzeni

4. Yığın (Stack):

- Yığın, bir fonksiyon çağrıldığında kaydedilen bilgilerle birlikte otomatik değişkenlerin saklandığı yerdir.
- Bir fonksiyon her çağrıldığında, geri dönülecek yerin adresi ve arayanın ortamı ile ilgili bazı makine kayıtları gibi bazı bilgiler yığında saklanır.
- Yeni çağrılan fonksiyon daha sonra otomatik ve geçici değişkenleri için yığında yer ayırır.
- Bu, C'deki özyinelemeli fonksiyonların çalışmasını sağlar.
- Özyinelemeli bir işlev kendisini her çağırdığında, yeni bir yığın çerçevesi kullanılır, bu nedenle bir değişken kümesi, işlevin başka bir örneğindeki değişkenlere müdahale etmez.

C Programlarının Bellek Düzeni

5. Öbek (Heap): ►Öbek, dinamik bellek ayırma işleminin genellikle gerçekleştiği bölümdür.

►Öbek alan malloc, realloc ve free tarafından yönetilir.

```
#include <stdio.h>

int global; /* Uninitialized variable stored in bss*/
int main(void)
{
    int *ptr_one;
    ptr_one = (int *)malloc(sizeof(int)); /* memory allocating in heap segment */
    int c;//local variable stored in stack
    static int i = 100; /* Initialized static variable stored in DS*/
    static int k; /* Initialized static variable stored in bss*/
    return 0;
}
```

Geniş Programlar Oluşturma

- Büyük programlar genelde farklı klasörlerde bulunan .h uzantılı headerdosyaları ve .c uzantılı dosyalardan oluşur.
- Preprocessor programı #include<"dosya_adı"> direktifini aldığında bu dosyayı aynı klasörde veya sistemde tanımlı yerlerde arar.
- Bulamaz ise hata mesajı verilir derleme durdurulur.
- .h uzantılı dosyalarda, #include, #define direktifleri, Struct yapılar, fonksiyon prototipleri bulunabilir.

Geniş Programlar Oluşturma

dosya1.c

```c
#include "pgm.h"

int main(void) {
    int i;
    for (i = 0; i < N; i++) {
        f2();
    }
    return 0;
}
```

dosya2.c

```c
#include "pgm.h"

void f2(void) {
    printf("Hello from f2()\n");
}
```

```
Hello from f2()
Hello from f2()
Hello from f2()
Hello from f2()
Hello from f2()
```

Dosya1 çalıştırılırsa çıktı.

pgm.h

```c
#ifndef PGM_H
#define PGM_H

#include <stdio.h>

#define N 5

/* Prototype only */
void f2(void);

#endif
```

Özyineleme (Rekürsif)

- Kendi kendini çağıran fonksiyonlardır.
- Eğer fonksiyon temel durum ile çağırılırsa bir sonuç döndürür.
- Eğer fonksiyon daha karmaşık bir problem ile çağırılırsa, fonksiyon problemi iki kavramsal parçaya böler;
  - Birincisi: fonksiyonun işi nasıl yapacağını bildiği kısım
  - İkincisi: fonksiyonun işi nasıl yapacağını bilmediği kısım
    - İkinci kısım orijinal probleme benzemelidir.
    - Fonksiyonun bilmediği kısmı çözebilmek için kendisinin bir kopyasını çalıştırır.
- Sonunda temel durum çözülür.

Özyineleme (Rekürsif)

- 1'den N sayısına kadar olan sayıları ekrana yazdıran program.

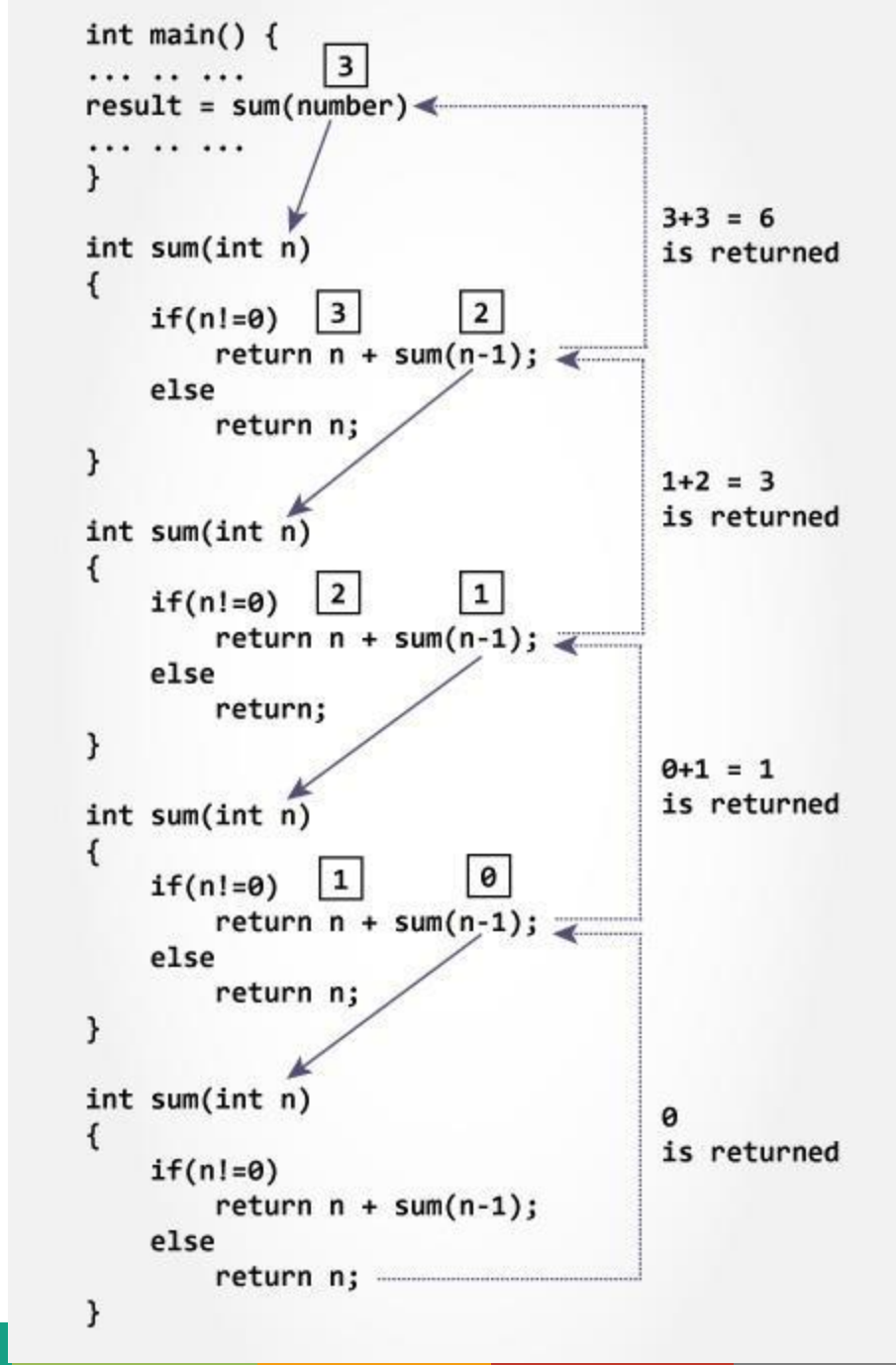
```c
#include <stdio.h>
int print_N(int n) {
    if (n == 0)
        return 0;
    print_N(n - 1);
    printf("%d\n", n);

}
 int main(void) {
    int sayi= 10;
    print_N(sayi);
    return 0;
}
```

Özyineleme (Rekürsif)

- 1'den N sayısına kadar olan sayıların toplamını bulan rekürsif bir fonksiyon tasarlamak istersek.

```
#include <stdio.h>
int topla(int n) {

        if(n == 1)
                return n;
        else
                return(n + topla(n -1));
} int main(void) {


        int sayi = 10;

        printf("Sonuc= %d", topla(sayi));
        return 0;

}
```

Özyineleme (Rekürsif)



Özyineleme (Rekürsif)

- Rekürsif olarak çarpım tablosunu yazdıran program.

```
#include <stdio.h> void
print_mult_table(x) {
        int i;
        if(x<=10) {
                for(i = 1; i<11; i++)
                        printf("%-3d", x*i);
                printf("\n");
                return print_mult_table(x + 1);
        }
        else return 1;
}

int main(void){
        int x = 1;
        print_mult_table(x);
        return 0;
}
```

Özyineleme (Rekürsif)

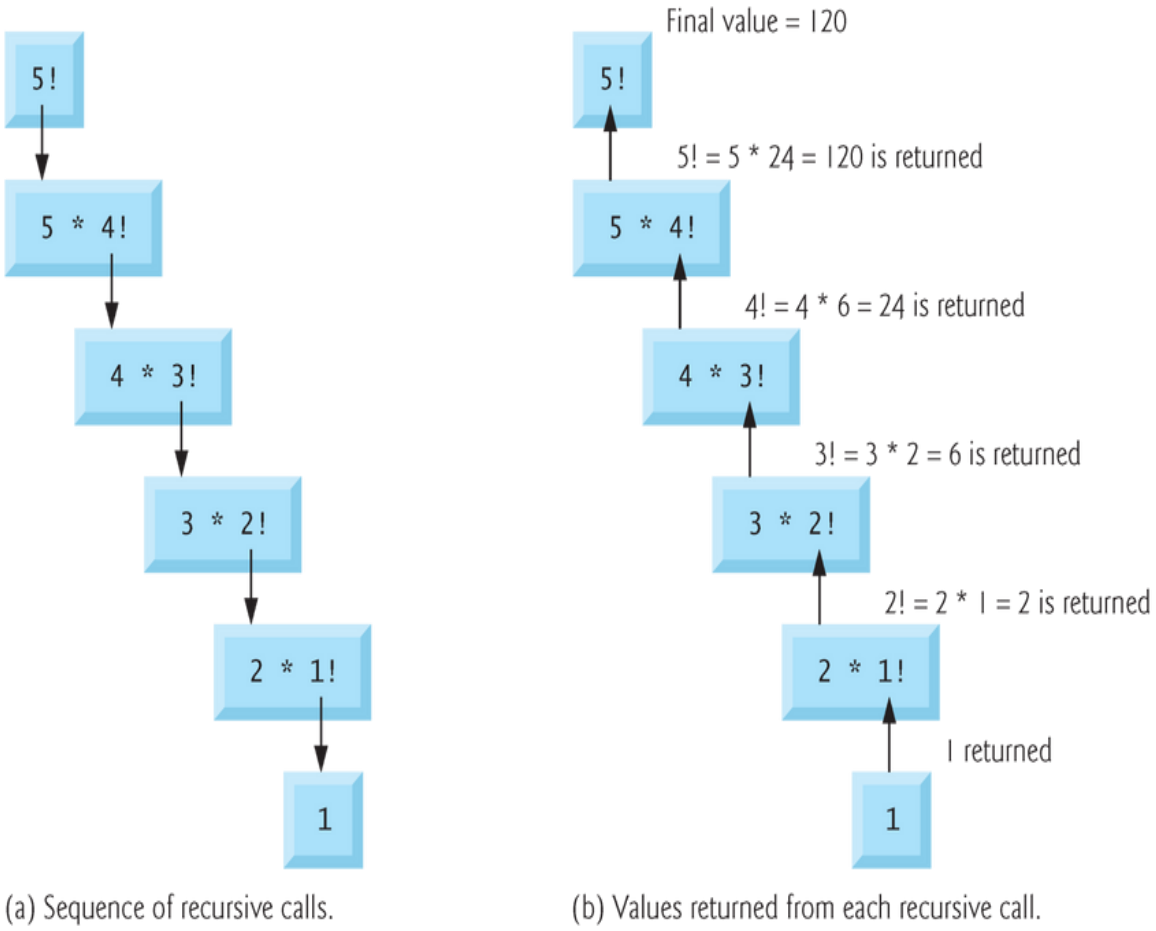
- Faktöriyel probleminin rekürsif tanımlaması aşağıdaki gibi yapılır.

n! = n. (n-1)!

- Örneğin: faktöriyel
  - 5! = 5. 4. 3. 2. 1
  - Dikkat edin
    5! = 5. 4!
    4! = 4. 3!...
  - Faktöriyel hesabı rekürsif olarak hesaplanabilir.
  - Temel durumu çöz (1! = 0! =1) daha sonra

    2! = 2. 1! = 2*1 = 2
    3! = 3. 2! = 3*2 = 6

Özyineleme (Rekürsif)



(a) Sequence of recursive calls.

(b) Values returned from each recursive call.

Özyineleme (Rekürsif)

- 0-10 arası tam sayılı faktöriyelleri hesaplamak ve yazdırmak için özyineleme programı.

```c
#include <stdio.h>

long faktoriyelHesapla(long n) {
    if (n <= 1) {
        return 1;
    } else {
        return n * faktoriyelHesapla(n - 1);
    }
}

int main(void) {
    int i;
    for (i = 0; i <= 10; i++) {
        printf("%d! = %ld\n", i, faktoriyelHesapla(i));
    }
    return 0;
}
```

Özyineleme Fibonacci Sayıları

- Fibonacci serisi: 0, 1, 1, 2, 3, 5, 8... Her bir sayı
- kendinden önceki iki sayının toplamıdır.
- Temel durum:

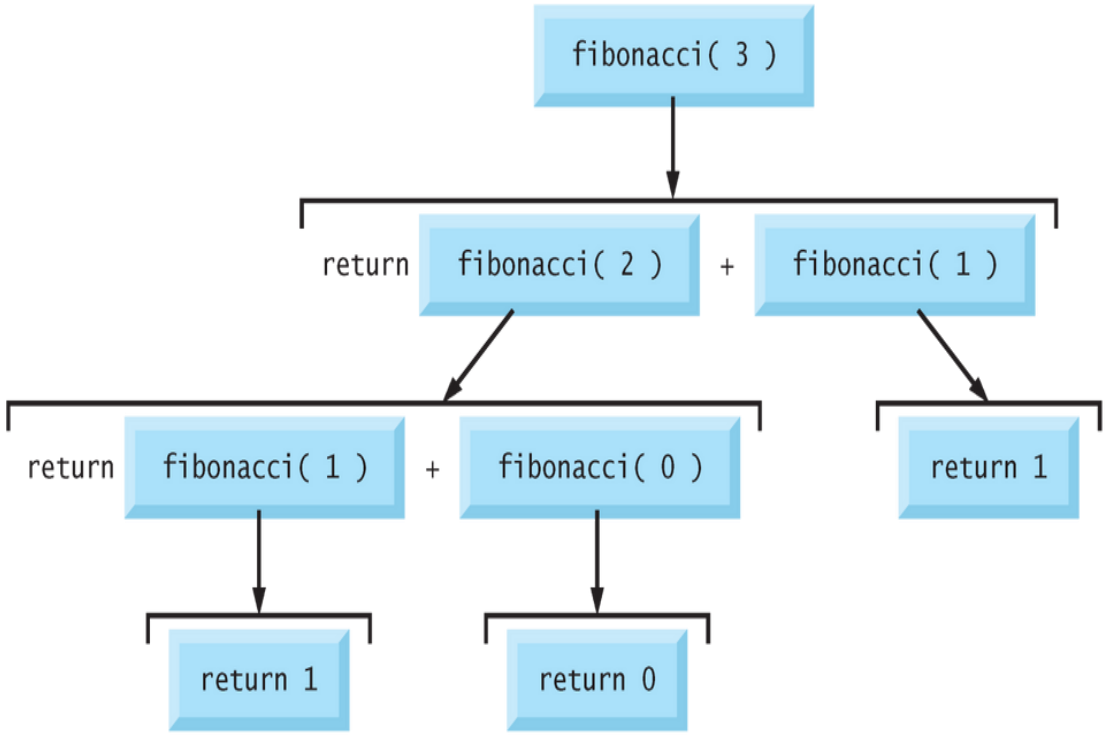
  Fib(0) = 0
  Fib(1) = 1

- Rekürsif olarak çözülebilir :

  Fib(n) = Fib(n-1) + Fib(n-2)

Özyineleme FibonacciSayıları

- Şekil, Fibonacci işlevinin fibonacci(3)'ü nasıl değerlendireceğini göstermektedir.

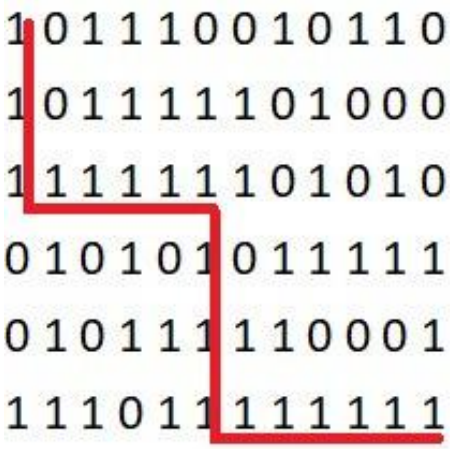


Özyineleme Fibonacci Sayıları

- İlk n adet Fibonacci sayısını yazdıran program

```c
#include <stdio.h>

long fibonacciHesapla(long n) {
    if (n == 0 || n == 1) {
        return n;
    } else {
        return fibonacciHesapla(n - 1) + fibonacciHesapla(n - 2);
    }
}

int main(void) {
    int i, n;
    printf("Kaç Fibonacci sayısı hesaplanacak?: ");
    scanf("%d", &n);
    for (i = 0; i < n; i++) {
        printf("Sayı %d: %ld\n", i, fibonacciHesapla(i));
    }
    return 0;
}
```

Rekürsif En Kısa Yol Bulma Ödevi



Ödev: Rekürsif Yol Bulma

Tanım:

m x n boyutunda, 1 ve 0'lardan oluşan bir matris verilmiştir.

- **1** değeri, geçilebilir yolu;
- **0** değeri, duvarı temsil etmektedir.

Matris içerisindeki hücreler, satır ve sütun indeksleri 0'dan başlayacak şekilde tanımlanmıştır. Başlangıç noktası **(0, 0)** iken, hedef nokta **(m-1, n-1)**'dir. Eğer başlangıç veya hedef noktası 0 ise, geçerli bir yol bulunamaz.

Görev:

0,0 noktasından (m-1, n-1) noktasına ulaşan **en kısa yolu** rekürsif olarak bulan bir C programı yazınız.

- Sadece dört yönde hareket edebilirsiniz: **Yukarı, Aşağı, Sola ve Sağa**.
- Rekürsif çözümde, ziyaret edilmiş hücreleri takip ederek sonsuz döngüye girmemeye dikkat ediniz.

Rekürsif En Kısa Yol Bulma Ödevi

- **Ek Gereksinimler:**

1. **Giriş:**
    - Matris boyutları (m ve n) ve matrisin kendisi kullanıcıdan alınabilir veya program içinde sabit olarak tanımlanabilir.
2. **Çıkış:**
    - Eğer geçerli bir yol bulunursa, program bulunan yolun uzunluğunu ekrana yazdırmalı ve tercihe bağlı olarak matristeki yolu (örneğin, yolu '*' ile işaretleyerek) görsel olarak göstermelidir.
    - Eğer geçerli bir yol bulunamazsa, "Yol bulunamadı" gibi uygun bir mesaj veriniz.

Rekürsif En Kısa Yol Bulma Ödevi

3. **Örnek:**

   Aşağıda örnek bir matris ve beklenen çıktıya ilişkin örnek verilmiştir:

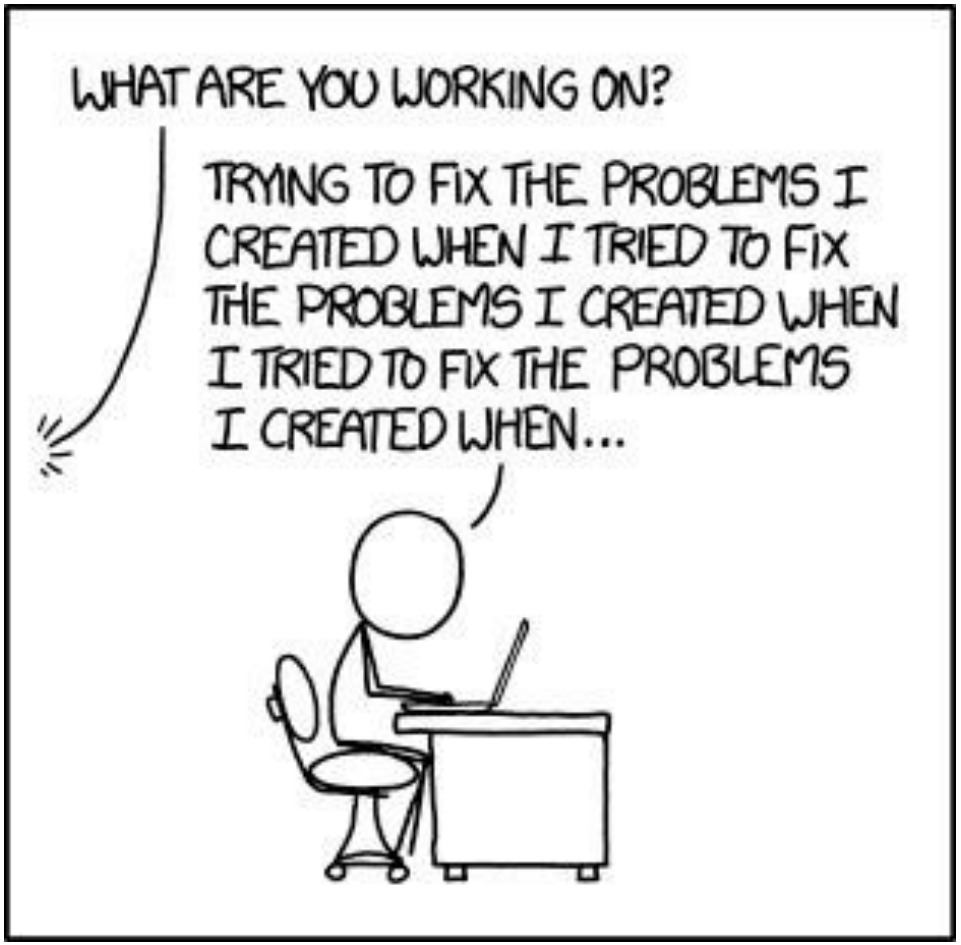
```
Örnek Matris (4x4):
1 1 1 0
0 1 1 1
1 1 0 1
1 1 1 1

Beklenen Çıktı (Örnek Yol):
* * * 0
0 * * *
1 * 0 *
1 * * *
Yol Uzunluğu: 7
```

Not:

- Rekürsif yaklaşım kullanılarak, her adımda geçerli dört yönü (Yukarı, Aşağı, Sola, Sağa) deneyerek çözümü bulunuz.
- Programınız, eğer bir yola ulaşamazsa veya başlangıç/varış noktası geçilemezse uygun hata mesajı göstermelidir.

Gelecek Hafta

- Biraz Mizah…



Kaynaklar

- **Doç. Dr. Caner ÖZCAN, KBÜ Yazılım Mühendisliği www.canerozcan.net***
- Doç. Dr. Fahri Vatansever, “Algoritma Geliştirme ve Programlamaya Giriş”, Seçkin Yayıncılık, 12. Baskı, 2015.
- Kaan Aslan, “A’dan Z’ye C Klavuzu8. Basım”, Pusula Yayıncılık, 2002.
- Paul J. Deitel, “C How toProgram”, HarveyDeitel.
- “A bookon C”, AllKelley, İraPohl

* Bu dersin slaytları genelde bu kaynaktan türetilmiştir.

S o r u l a r ?

Dinlediğiniz için teşekkürler