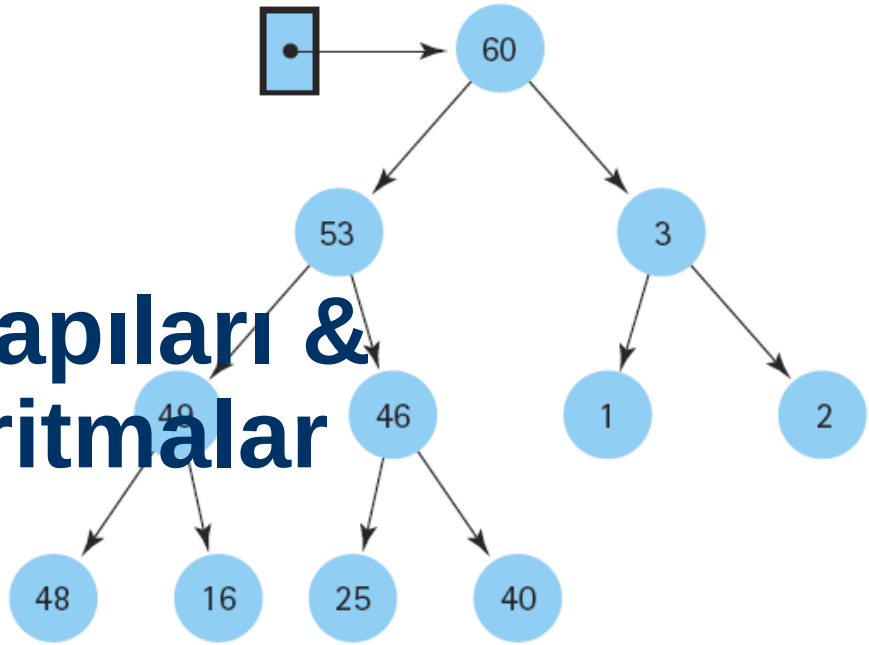
Veri Yapıları & Algoritmalar



İçerik

- Ders kitabı
- Puanlama
- Yazılım

Ders kitabı

- C & Data Structures
  - P. S. Deshpande, O. G. Kakde
  - CHARLES RIVER MEDIA, INC. Hingham, Massachusetts
- Veri Yapıları ve Algoritmalar
  - Dr. Rifat Çölkesen
  - Papatya Yayıncılık

Puanlama

- Vize sınavı (%30)
- Final sınavı (%40)
- Ödev ve proje (%30)

Yazılım: C/C++ editor

- **Dev C++**
  - **http://www.bloodshed.net/**

KISIM 0: GİRİŞ

- ***Veri yapıları nedir?***
  - ***Bir veri yapısı şöyle tanımlanabilir;***
    - ***(1) veri elemanlarının mantıksal açıdan düzenlenmesi ve***
    - ***(2) bu elemanlara erişmek için ihtiyaç duyduğumuz işlemlerin kümesi.***

Atomik Değişkenler

- Atomik değişkenler bir anda tek bir değer saklayabilen değişkenlerdir.

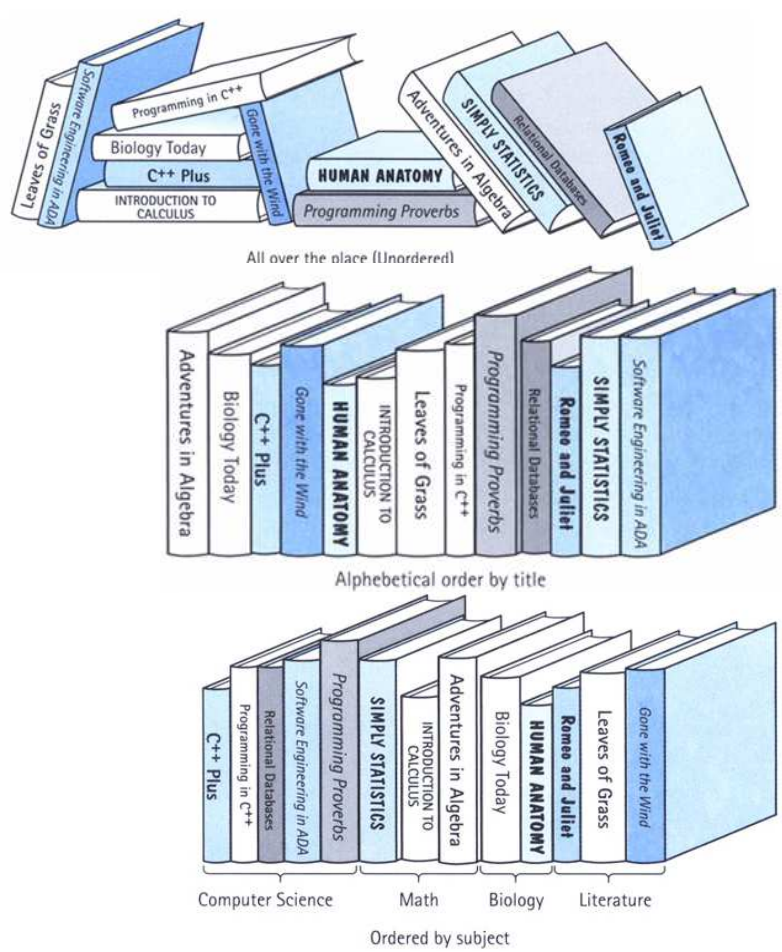
  int num;

  float s;

- Bir atomik değişkende tek bir değer saklanır ve alt parçalara ayrılamaz.

Veri Yapıları Nedir?

- Örnek:Kütüphane
  - Elemanlardan meydana gelir (books)
  - Bir kitaba ulaşabilmek için kitapların düzeni ile ilgili bilgiye ihtiyaç vardır.
  - Kullanıcılar kitaplara kütüphanecileri yardımıyla erişir.

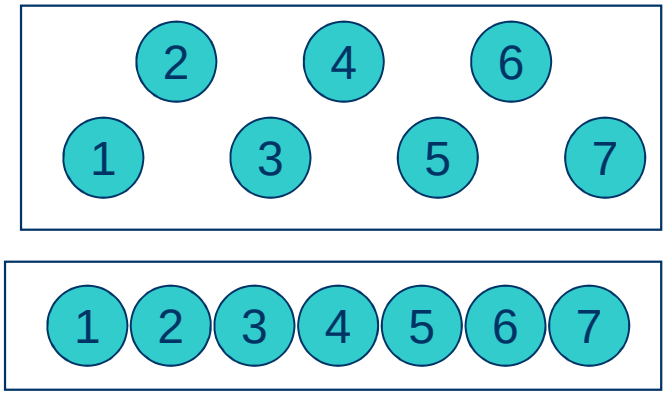


Temel veri yapıları

- Aşağıdaki yapıları içerir
  - Bağlı listeler (linked lists)
  - Yığıt ve kuyruk (Stack, Queue)
  - İkili ağaçlar (binary trees)
  - Ve diğerleri…

Algoritma nedir?

- **Algoritma:**
  - Hedeflenen sonucu elde etmek için adımların hesaplanabilir bir dizisidir.
  - Veri yapıları ile ilişkilidir
    - Örnek: bir elemanı bul



Özet

Algorithms + Data Structures = Programs

Algorithms $\leftrightarrow$ Data Structures

Kısım 0: C DİLİ

1. *Adresler (ADDRESS)*
2. *İşaretçiler (POINTERS)*
3. *Diziler (ARRAYS)*
4. *Bir dizideki her bir elemanın adresi (ADDRESS OF EACH ELEMENT IN AN ARRAY)*
5. *Gösterge kullanarak bir diziye erişim ve dizi elemanları ile çalışma (ACCESSING & MANIPULATING AN ARRAY USING POINTERS)*
6. *Gösterge kullanarak bir diziye erişim ve dizi elemanları ile çalışma – bir diğer durum (ANOTHER CASE OF MANIPULATING AN ARRAY USING POINTERS)*
7. *İki boyutlu dizi (TWO-DIMENSIONAL ARRAY)*
8. *Gösterge dizileri (POINTER ARRAYS)*
9. *Yapılar (STRUCTURES)*
10. *Yapı göstergeleri (STRUCTURE POINTERS)*

Kısım 0: C DİLİ

1. *Adres*

Her değişkenin iki özelliği vardır adres ve değer

3 adresiyle verilen değer 45.
2 adresiyle verilen değer "Dave"

1	4096
2	"Dave"
3	45
4	"Matt"
5	95.5
6	"wbru"
7	0
8	"zero"

```
cout << “y’nin değeri : " << y << "\n";
cout << “y’nin adresi : " << &y << "\n\n";
```

Kısım 0: C DİLİ

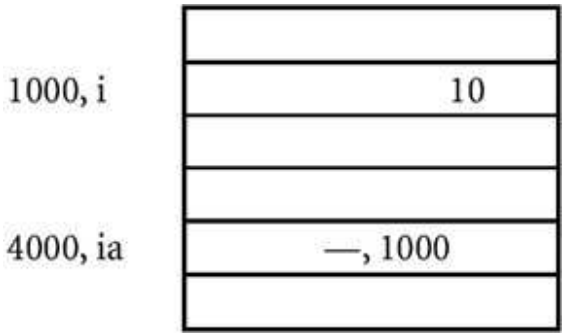
2. İşaretçiler (POINTERS)

1. ***Değeri aynı zamanda bir adres olan bir değişkendir.***
2. ***Bir tamsayının adresini tanımlayan bir işretçi de bir tamsayı değişkendir.***

ia: değişkenin değeri (1000)

&ia: ia değişkeninin adresi (4000)

***ia, ia ile verilen adresteki değişkenin değeri (10)**



Kısım 0: C DİLİ

```
int i;        //A
  int * ia;      //B
   cout<<“i değişkeninin adresi "<< &i << " değeri="<<i <<endl;
   cout<<“ia değişkeninin adresi " << &ia << " değeri= " << ia<< endl;

  i = 10;        //C
  ia = &i;   //D

            cout<<“değer atamanın ardından:"<<endl;
 cout<<“i değişkeninin adresi "<< &i << " değeri="<<i <<endl;
 cout<<“ia değişkeninin adresi" << &ia << " değeri= " << ia<< " point to: "<<
*ia;
```

Kısım 0: C DİLİ

Hatırlanması gereken noktalar

- **İşaretçiler bir değişkenin değerine dolaylı erişim için kolaylık sağlarlar.**
- **Değişken adından önce * operatörü yardımıyla işaretçi tanımlanabilir.**
- **& operatörü kullanılarak saklı bir değişkenin adresi alınabilir.**

Kısım 0: C DİLİ

3. DİZİLER (ARRAYS)

1. ***Dizi, bir veri yapısıdır***
2. ***Eleman sayısı bilinen durumlarda, aynı veri tipine ait birden çok eleman ile çalışmak için kullanılır.***
3. ***Bir dizi bileşik bir veri yapısıdır; tamsayılar gibi basit veri tiplerinin bir araya gelmesinden oluşur.***

1. ***int a[5];***
2. ***for(int i = 0;i<5;i++)***
    1. ***{a[i]=i; }***

Kısım 0: C DİLİ

4. BİR DİZİDEKİ BİR ELEMANIN ADRESİ

Dizinin her elemanının bir bellek adresi vardır.

```
void printdetail(int a[])
{
    for(int i = 0;i<5;i++)
    {
       cout<< “dizideki değer “<< a[i] <<“ onun adresi: “ << &a[i]);
    }
```

Kısım 0: C DİLİ

5. İŞARETÇİ KULLANARAK BİR DİZİYE ERİŞİM& İŞLEME

- *Bir işaretçi kullanarak da bir diziye erişilebilir.*
- *Eğer bir dizi tamsayıları saklıyorsa işaretçiler de tamsayı olmalıdır .*

```
void printarr_usingpointer(int a[])
{
    int *pi;
    pi=a;
     for(int i = 0;i<5;i++)
      {
         cout<<"value array:" << *pi << " address: "<< pi<<endl;
       pi++;
      }
}
```

Kısım 0: C DİLİ

6. İŞARETÇİ KULLANARAK BİR DİZİYİ İŞLEME İÇİN BİR DİĞER DURUM

Dizi adı bir işaretçi sabitidir : programda onun değeri değiştirilemez.

int a[5]; int *b;

a=b; //error

b=a; //OK

```
void print_usingptr_a(int a[])
{

    for(int i = 0;i<5;i++)
      {cout<<"a[" << i << "]=" <<*a<<endl;
        a++;}
}
```

Kısım 0: C DİLİ

7. İKİ BOYUTLU DİZİ

int a[3][2];

3	1
5	2
8	7

```
int a[3][2];
for(int i = 0;i<3;i++)
    for(int j=0;j<2 ;j++)
    {
            {
                    a[i][j]=i+j+i*j;
            }
    }
```

```
void print_usingptr(int a[][2])
{
    int *b;
    b=a[0];
    for(int i = 0;i<6;i++)
    {
        cout<<"value :" << *b<<" in address: "<<b<<endl;
        b++; }
}
```

Kısım 0: C DİLİ

8. İŞARETÇİ DİZİLERİ

- *Bir işaretçi dizisi tanımlanabilir (tamsayı dizilerine benzer şekilde).*
- *İşaretçi dizisinde, dizi elemanları tamsayı değerlerine gösteren işaretçileri saklar.*

```
int *a[5];
main()
{    int i1=4,i2=3,i3=2,i4=1,i5=0;
     a[0]=&i1;
     a[1]=&i2;
     a[2]=&i3;
     a[3]=&i4;
     a[4]=&i5;

     printarr(a);
     printarr_usingptr(a);
}
```

```
void printarr(int *a[])
{
    printf("Address1\tAddress2\tValue\n");
    for(int  j=0;j<5;j++)
    {
        cout<<*a[j]<<"\t"<<a[j]<<"\t"<<&a[j]<<endl;
    }
}
```

Kısım 0: C DİLİ

9. STRUCTURES

- ***Yapılar farklı veri tiplerinde birçok eleman söz konusu olduğunda kullanılır.***
- ***Fakat siz bu veriye tek bir varlık olarak referans verebilirsiniz.***
- ***Erişim verisi:***

Yapıadı.elemanadı

```
struct student
{
    char name[30];
    float marks;
};

main ( )
{
    student student1;
    char s1[30];float f;
    cin>>student1.name;
    cin>>student1.marks;

    cout<<"Name is:"<<student1.name;
    cout<< "Marks are:" << student1.marks;
}
```

Kısım 1: C DİLİ

10. STRUCTURE POINTERS

Bir yapı işaretçisi ile yapı işlenebilir

```
struct student
{   char name[30];
    float marks;    };
main ( )
{
     struct student *student1;
     struct student student2;
     student1 = &student2;
     cin>>student1->name; // cin>>(*student1).name;
     cin>>student2.marks;
     cout<<(*student1).name<<" : " << student2.marks;
}
```

Kısım 2: **FUNCTION & RECURSION**

- 1. FONKSİYON
- 2. YIĞIT KAVRAMI (THE CONCEPT OF STACK)
- 3. BİR FONKSİYON ÇAĞRISI SIRASINDA ÇALIŞTIRMA SIRASI
- 4. PARAMETRE GEÇİRME (PARAMETER PASSING)
- 5. REFERANS İLE ÇAĞIRMA (CALL BY REFERENCE)
- 6. DEĞİŞKEN REFERANSLARI ÇÖZÜMLEME
- 7. ÖZYİNELEME (RECURSION)
- 8. ÖZYİNELEMEDE YIĞIT TAŞMASI
- 9. BİR ÖZYİNELEMELİ FONKSİYON YAZMA
- 10. ÖZYİNELEME TİPLERİ

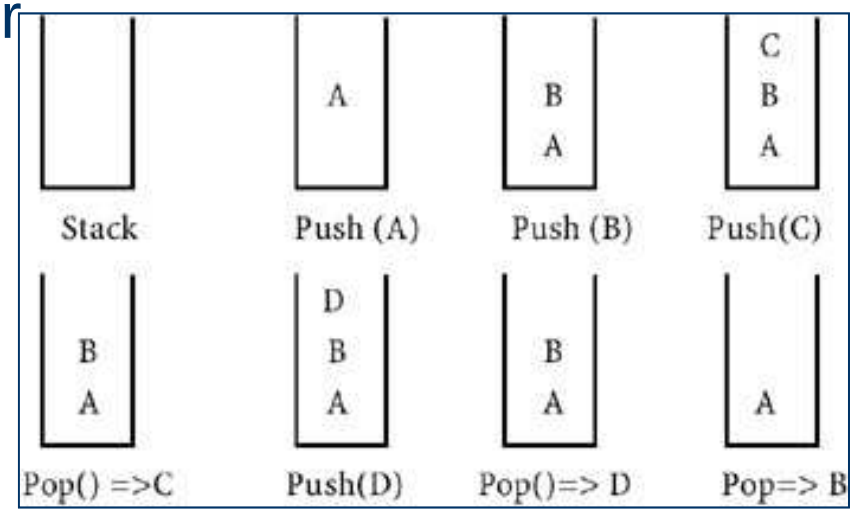
Kısım 2: **FUNCTION & RECURSION**

- 1. FUNCTION
  - Yazılım için modülerlik sağlar
  - Karmaşık görevleri yönetilebilir, küçük görevlere böler
  - İş tekrarını önler

```
int add (int x, int y)
{
    int z;
    z = x + y;
    return z;
}
```

Kısım 2: **FUNCTION & RECURSION**

- 2. YIĞIT (STACK) KAVRAMI
  - Bir yığıt, değerlerin “son giren ilk çıkar” mantığına uygun olarak saklandığı ve geri alındığı ve bu işlemleri **push** ve **pop** operasyonları ile yerine getiren bir bellek türüdür.



Kısım 2: **FUNCTION & RECURSION**

- 3. BİR FONKSİYON ÇAĞRISI KULLANIRKEN ÇALIŞTIRMA SIRASI
  - Fonksiyon çağrıldığında, geçerli çalıştırma geçici olarak durdurulur ve kontrol çağrılan fonksiyona geçer. Çağrıdan sonra çalıştırma kaldığı yerden devam eder.
  - Hangi noktaya geri dönüleceği, bir sonraki komutun adresi yığıtta saklanır. Fonksiyon çağrısı sona erdiğinde, yığıtın en üstündeki elemanın adresi alınır.

Kısım 2: **FUNCTION & RECURSION**

- 3. BİR FONKSİYON ÇAĞRISI KULLANIRKEN ÇALIŞTIRMA SIRASI
  - Fonksiyonlar veya alt programlar bir yığıt kullanılarak geliştirilir.
  - Bir fonksiyon çağrıldığında, bir sonraki komutun adresi yığıta push edilir.
  - Fonksiyon sona erdiğinde, çalıştırma için adres pop operasyonu ile alınır.

Kısım 2: **FUNCTION & RECURSION**

- 3. FONKSİYON ÇAĞRIS KULLANIMI SIRASINDA ÇALIŞTIRMA DİZİSİ
- Sonuç:?

```
main ( )
{
     printf ("1 \n");
     printf ("2 \n");
     f1();
     printf ("3 \n");
     printf ("4 \n");
}
void f1()
{
     printf ("f1-5 \n");
     printf ("f1-6 \n");
     f2 ( );
     printf ("f1-7 \n");
     printf ("f1-8 \n");
}
void f2 ()
{
     printf ("f2-9 \n");
     printf ("f2-10 \n");
}
```

Kısım 2: **FUNCTION & RECURSION**

- 4. PARAMETER * REFERENCE PASSING
  - *Değerle çağırma*
    - Çağrı öncesi ve sonrası değer aynı kalır
  - *Referansla çağırma*
    - *Fonksiyon bitiminde değer değişir*

```
void (int *k)
{
     *k = *k + 10;
}
```

```
void (int &k)
{
     k = k + 10;
}
```

Kısım 2: **FUNCTION & RECURSION**

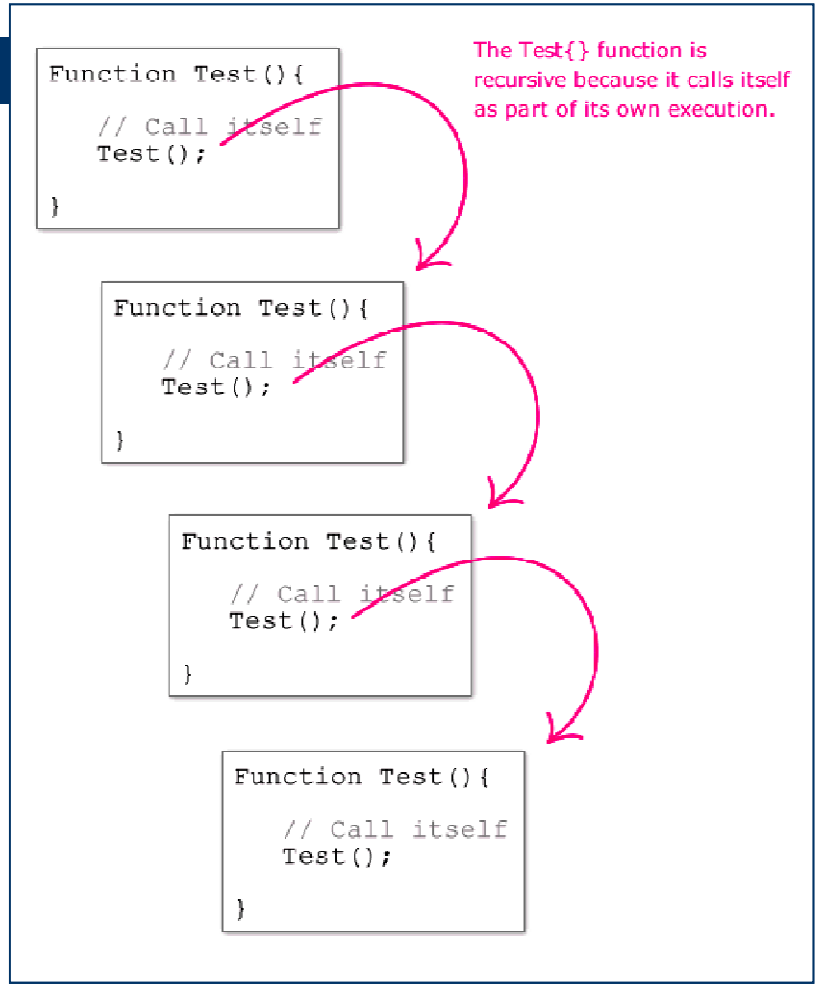
- 6. DEĞİŞKEN REFERANSLARI ÇÖZÜMLEME

Bir değişken çoklu referanslar kullanılarak çözülebilir, yerel tanımlama daha fazla referansa verilir

```
int i =0;  //Global variable
main()
{
    int i ;   // local variable for main
    void f1(void) ;
    i =0;
    cout<<"value of i in main: "<< i <<endl;
    f1();
    cout<<"value of i after call:" << i<<endl;
}
void f1()
{
    int i=0;   //local variable for f1
    i = 50;
}
```

Kısım 2: **FUNCTION & RECURSION**

- 7. RECURSION
  - Doğrudan veya dolaylı olarak bir fonksiyonun kendini çağırması metoduna verilen isimdir.
  - Problems: özyinelemenin durması?



Kısım 2: **FUNCTION & RECURSION**

- 7. RECURSION

Example: Factorial

- The factorial is defined in this way:

$$n! = \begin{cases} 1 & n = 0 \\ 1 \times 2 \times \cdots \times n & n > 0 \end{cases}$$

This can be computed by a loop.

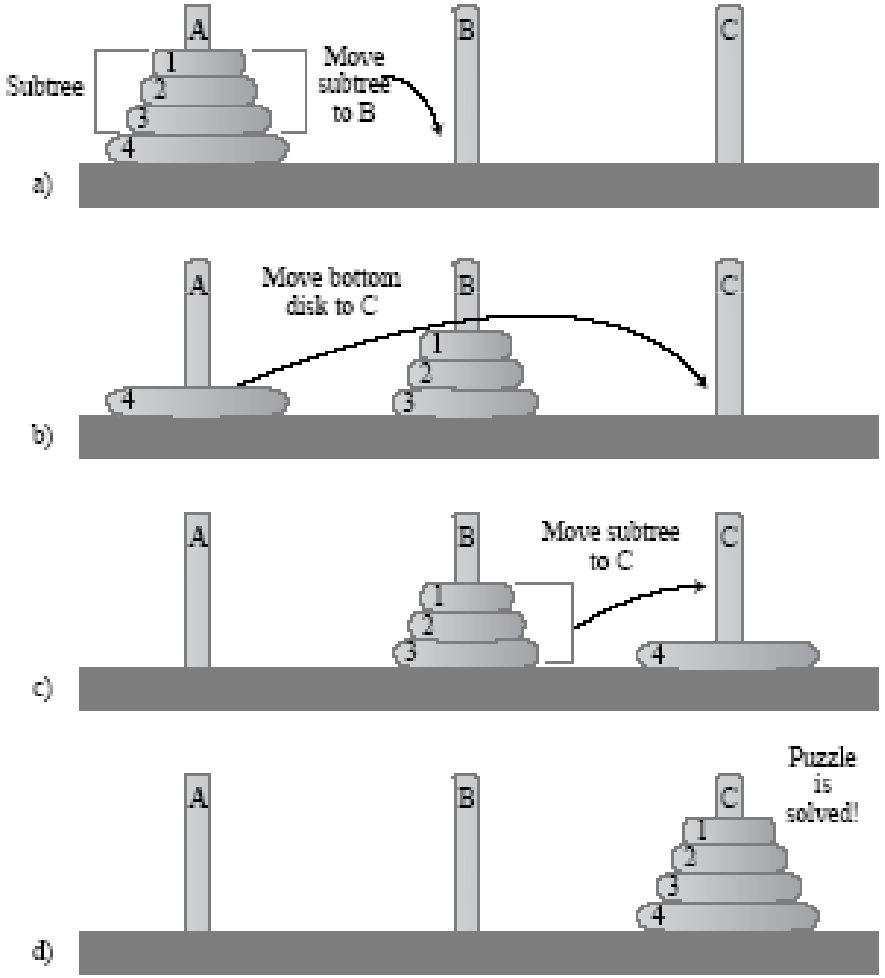
- We can also define the factorial as:

$$n! = \begin{cases} 1 & n = 0 \\ n \times (n-1)! & n > 0 \end{cases}$$

So, if we know how to compute the factorial of $n-1$, we know how to compute the factorial of n.

Kısım 2: **FUNCTION & RECURSION**

- 7. RECURSION: Hanoi tower



Kısım 2: **FUNCTION & RECURSION**

- 7. RECURSION

Example: Factorial

```
int factorial(int n)  // assumes n >= 0
{
  if (n == 0)
    return 1;
  else
    return n * factorial(n-1);
}
```

To see how the computation is done, trace `factorial(3)`:

```
factorial(3) = 3 * factorial(2)
             = 3 * (2 * factorial(1))
             = 3 * (2 * (1 * factorial(0)))
             = 3 * (2 * (1 * 1))
```

Kısım 2: **FUNCTION & RECURSION**

- 9. BİR RECURSIVE FUNCTION YAZMA
  - Özyineleme bize doğal yoldan bir programı yazma imkanı verir. Yığıt kullanımı nedeniyle bir özyinelemeli program daha yavaş çalışır.
  - Bir özyinelemeli programda özyineleme, sonlanma şartları ile belirlemelisiniz.

Kısım 2: **FUNCTION & RECURSION**

- 10. ÖZYİNELEME TİPLERİ
  - **DOĞRUSAL RECURSION**
  - **KUYRUK RECURSION**
  - **İKİLİ RECURSION**
  - **ÜSSEL RECURSION**
  - **GÖMÜLÜ RECURSION**
  - **SIRADIŞI RECURSION**

Kısım 2: **FUNCTION & RECURSION**

- 10. ÖZYİNELEME TİPLERİ
  - **DOĞRUSAL ÖZYİNELEME**
    - **Her seferinde tek bir çağrı kendi kendine fonksiyon çalıştırır.**

```
int factorial (int n)
{
 if ( n == 0 )
   return 1;
 return n * factorial(n-1);
}
```

Kısım 2: **FUNCTION & RECURSION**

- 10. ÖZYİNELEME TİPLERİ
  - **KUYRUK RECURSION**
    - **Doğrusal özyinelemenin bir formudur.**
    - **Bu türde, yineleme çağrısı metodun en sonunda yapılır. Sıklıkla yineleme çağrısının değeri döndürülür.**

```
int gcd(int m, int n)
{
        int r;
        if (m < n) return gcd(n,m);
        r = m%n;
        if (r == 0) return(n);
        else return(gcd(n,r));
}
```

Kısım 2: **FUNCTION & RECURSION**

- 10. ÖZYİNELEME TİPLERİ
  - **İKİLİ RECURSION**
    - **Bazı yinelemeli fonksiyonların tek bir çağrısı yoktur, onlar bazen iki bazen daha fazla çağrı içerir.**

```
int choose(int n, int k)
{
        if (k == 0 || n == k) return(1);
        else return(choose(n-1,k) + choose(n-1,k-1));
}
```

Kısım 2: **FUNCTION & RECURSION**

- 10. ÖZYİNELEME TİPLERİ
  - **ÜSSEL RECURSION**
    - **Bir üssel özyinelemeli fonksiyonda eğer fonksiyon çağrılarının bir sunumunu ortaya koymak istiyorsanız, veri setindeki ilişkilerin sayısı üssel olacaktır.**
    - **(exponential meaning if there were *n* elements, there would be $O(a^n)$ function calls where a is a positive number)**

Kısım 2: **FUNCTION & RECURSION**

- 10. ÖZYİNELEME TİPLERİ
  - **ÜSSEL RECURSION**

```
void print_array(int arr[], int n)
{
        int i;
        for(i=0; i<n; i++) printf("%d ", arr[i]);
        printf("\n");
}

void print_permutations(int arr[], int n, int i)
{
        int j, swap;
        print_array(arr, n);
        for(j=i+1; j<n; j++) {
                swap = arr[i]; arr[i] = arr[j]; arr[j] = swap;
                print_permutations(arr, n, i+1);
                swap = arr[i]; arr[i] = arr[j]; arr[j] = swap;
        }
}
```

Kısım 2: **FUNCTION & RECURSION**

- 10. ÖZYİNELEME TİPLERİ
  - **İÇ İÇE RECURSION**
    - **İç içe yinelemede, argümanlardan birisi yinelemeli fonksiyonun kendisidir.**
    - **Bu fonksiyonlar hızlı şekilde büyümeye meyillidir.**

```
int ackerman(int m, int n)
{
        if (m == 0) return(n+1);
        else if (n == 0) return(ackerman(m-1,1));
        else return(ackerman(m-1,ackerman(m,n-1)));
}
```

Kısım 2: **FUNCTION & RECURSION**

- 10. YİNELEME TİPLERİ
  - **SIRADIŞI RECURSION**
    - Bir yineleme fonksiyonu her zaman kendi kendini çağırmaya ihtiyaç duymaz.
    - Bazı yinelemeli fonksiyonlar çiftler halinde veya daha büyük gruplar halinde çalışır. Örneğin, A fonksiyonu B fonksiyonunu çağırır, B fonksiyonu C fonksiyonunu çağırır ve en sonunda C fonksiyonu A fonksiyonunu çağırır.

Kısım 2: **FUNCTION & RECURSION**

- 10. YİNELEME TİPLERİ
  - **SIRADIŞI RECURSION**

```
int is_even(unsigned int n)
{
        if (n==0) return 1;
        else return(is_odd(n-1));
}

int is_odd(unsigned int n)
{
        return (!is_even(n));
}
```

Alıştırma 1: Recursion

```
*
**
***
****
*****
******
*******
********
```

Alıştırma 2: Recursion

- H10->H2 çevrimini yapınız