2

Özyineleme (recursion)

Kendi kendini çağıran fonksiyonlara özyineli (recursive) fonksiyon denilir. Özyineli fonksiyonlar, ileri bilgisayar uygulamalarında çok kullanılır. Bilgisayar biliminin zor sayılan konularından birisidir. Çoğunlukla, döngülerle çözülebilen problemler, özyineli fonksiyonlarla çok daha kolay olarak çözülebilir.

Özyineleme eylemi, problemi, en üstten başlayarak adım adım daha küçük parçalarına ayırır. Daha çok parçalanamayn en basit parçasına *taban parçası* denilir. Özyineleme en üstten başlayarak tabana kadar iner. Sonra tabanda bulduğu değeri geriye doğru en üste kadar taşır. Bu niteliği, döngülerden farklıdır.

Bu bölümde, bilmeyenlere konuyu açıklayabilmek için, bazı basit örnekler vermekle yetineceğiz.

Çarpınım (factoriel) problemini daha önce döngülerle çözmüştük. Program 2.1, bir sayının çarpımını özyineli fonksiyonla bulmaktadır. Bu yöntemin döngü ile yaptığımız çözümlere göre daha kısa ve daha anlaşılır olduğu apaçıktır.

Program 2.1.

```
1 #include <stdio.h>

  int func(int num)
  {
      int res = 0;
6
      if(num <= 0)
      {
          printf("\n hata! \n");
```

```
    }
    else if(num == 1)
    {
        return num;
    }
    else
    {
        res  = num * func(num -1);
        return res;
    }

    return -1;

}

int main()
{
    int num = 5 ;
    int fact  = func(num);

    if (fact > 0)
        printf("\n[%d] nin çarpınımı:  [%d]\n", num, fact);

     return 0;
}
```

Çarpınım, döngülerle ve özyineli fonksiyonla çözülebilen klasik bir örnektir. Özyinelemenin nasıl çalıştığını açıklamak için, çarpınım probleminde ana bellekte neler olup bittiğini açıklamak yararlı olabilir.

Çarpınım (factoriel) fonksiyonu şöyle tanımlanır.

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

4 sayısının çarpınımı bulunurken, özyineli fonksiyon şu işleri yapar:

```
f(4)        = 4 * f(3)
            = 4 * (3 * f(2))
            = 4 * (3 * (2 * f(1)))
            = 4 * (3 * (2 * (1 * f(0))))
            = 4 * (3 * (2 * (1 * 1)))
            = 4 * (3 * (2 * 1))
            = 4 * (3 * 2)
            = 4 * 6
            = 24
```

Çözümü dikkatle izlersek, ana bellekte yapılan işleri görebiliriz. f(4) hesaplanırken program akışı, sırasıyla, f(4) –> f(3) –> f(2) –> f(1) –> f(0) değerlerini buluyor. f() özyineleli fonksiyonunun taban parçası f(0) dır. Sonra, program akışı bunların değerleri olan 1 –> 1 –> 2 –> 6 –> 24 değerlerini kullanarak en başa kadar dönüyor.

Program 2.2 örneği, verilen bir sayıya kadar olan tam sayıların toplamını özyineli fonksiyonla bulmaktadır.

Program 2.2.

```
#include <stdio.h>

int count = 1;

void func(int sum)
{
    sum  = sum + count;
    count ++;

    if(count <= 9)
    {
        func(sum);
    }
    else
    {
        printf("\nSum is [%d] \n", sum);
    }

    return;
}

int main(void)
{
    int sum = 0;
    func(sum);
    return 0;
}
```

Program 2.3.

```
#include<stdio.h>

int Fibonacci(int);

main()
{
   int n, i = 0, c;

   scanf("%d",&n);

   printf("Fibonacci sayıları\n");

   for ( c = 1 ; c <= n ; c++ )
   {
      printf("%d\n", Fibonacci(i));
      i++;
   }

   return 0;
}
```

```
int Fibonacci(int n)
{
   if ( n == 0 )
      return 0;
   else if ( n == 1 )
      return 1;
   else
      return ( Fibonacci(n-1) + Fibonacci(n-2) );
}
```

```
/**
1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987, 1597,
 2584, 4181, 6765, 10946

Kaynak Linki : http://www.matematikciler.org/matematiksel-guzellikler
 /fibonacci-sayi-dizisi.html
*/
```

Program 2.4 örneği, verilen bir tam sayıyı ingilizce okunuşu biçiminde yazıyor. Bu problem iki yöntemle çözülebilir. Birincisi array kullanmaktır. İkinci yöntem ise özyineli fonksiyon kullanmaktır.

Problem çözülürken önce sayaklar ters sırada üretiliyor; yani basamak değeri daha küçük olan sayak daha önce bulunuyor. Önce birler basamağı, sonra onlar, sonra yüzler, vb. Ama sayı yazılırken ters sıra izleniyor. Önce basamak değeri büyük sayılardan başlanıyor, azalarak birler basamağına iniliyor.

Program 2.4.

```
#include <stdio.h>

 /* printd:  print n in decimal */
 void printd(int n)
 {
     if (n < 0) {
         putchar('-');
         n = -n;
     }
     if (n / 10)
         printd(n / 10);
     putchar(n % 10 + '0');
 }
```

```
/**

*/
```

Program 2.5 örneğinde carp() adlı özyineli fonksiyon ile iki sayının çarımı yapılmaktadır.

Program 2.5.

```c
#include<stdio.h>

int carp(int,int);

int main(){

    int a,b,product;
    printf("İki sayı giriniz: ");
    scanf("%d%d",&a,&b);

    product = multiply(a,b);

    printf("Girilen sayıların çarpımı: %d",product);

    return 0;
}

int carp(int a,int b){

    static int product=0,i=0;

    if(i < a){
        product = product + b;
        i++;
        carp(a,b);
    }

    return product;
}
```

```
/**
İki sayı giriniz: 5 9
Girilen sayıların çarpımı: 45
*/
```

Program 2.6 örneğinde ekok() adlı özyineli fonksiyon ile iki sayının en küçük ortak katını buluyor.

Program 2.6.

```c
#include<stdio.h>

int ekok(int,int);

int main(){

    int a,b,l;
    printf("İki pozitif tam sayı giriniz:  ");
    scanf("%d%d",&a,&b);

    if(a>b)
    l = ekok(a,b);
    else
        l = ekok(b,a);

    printf("Girilen sayılar için ekok: %d",l);
```

```
    return 0;
}

int ekok(int a,int b){

   static int temp = 1;

    if(temp % b == 0 && temp % a == 0)
        return temp;
    temp++;
    ekok(a,b);

   return temp;
}
```

```
/**
İki pozitif tam sayı giriniz: 2  5
Girilen sayılar için ekok : 10
*/
```

Program 2.7 örneğinde ebob() adlı özyineli fonksiyon ile iki sayının en büyük ortak bölenini buluyor.

Program 2.7.

```
#include<stdio.h>
int main(){
  int n1,n2,ebob;
  printf("\nİki tam sayı giriniz ");
  scanf("%d %d",&n1,&n2);
  ebob=ebobBul(n1,n2);
  printf("\n%d ile %d sayıları için ebob: %d",n1,n2,ebob);
  return 0;
}

int ebobBul(int x,int y){
     while(x!=y){
          if(x>y)
              return ebobBul(x-y,y);
          else
              return ebobBul(x,y-x);
     }
     return x;
}
```

```
/**
İki tam sayı giriniz: 6  15
6 mile 15 sayılarının ebob: 3
*/
```

Program 2.8 örneğinde asal() adlı özyineli fonksiyon ile verilen bir sayının asal olup olmadığını buluyor.

Program 2.8.

```c
#include<stdio.h>

int isPrime(int,int);

int main(){

    int num,prime;

    printf("Bir pozitif tam sayı giriniz: ");
    scanf("%d",&num);

    prime = isPrime(num,num/2);

   if(prime==1)
        printf("%d asaldır",num);
   else
      printf("%d asal değildir",num);

   return 0;
}

int isPrime(int num,int i){

    if(i==1){
        return 1;
    }else{
       if(num%i==0)
         return 0;
       else
         isPrime(num,i-1);
    }
}
```

```
/**
Pozitif bir tam sayı giriniz: 7
7 asaldır.
*/
```

Program 2.9 örneğinde asal() adlı özyineli fonksiyon ile verilen bir sayının asal olup olmadığını buluyor.

Program 2.9.

```c
#include<stdio.h>

long toBinary(int);

int main(){

    long binaryNo;
    int decimalNo;

    printf("Bir tam sayı giriniz: ");
    scanf("%d",&decimalNo);

    binaryNo = toBinary(decimalNo);
```

```
    printf("Binary karşılığı: %ld",binaryNo);

    return 0;
}

long toBinary(int decimalNo){

    static long binaryNo,remainder,factor = 1;

    if(decimalNo != 0){

        remainder = decimalNo % 2;
        binaryNo = binaryNo + remainder * factor;
        factor = factor * 10;
        toBinary(decimalNo / 2);
    }

    return binaryNo;
}
```

```
/**
Bir tam sayı giriniz: 11
1011
*/
```

Hanoi Kuleleri:

Program 2.10.

```
#include <stdio.h>

/*
K: kaynak,H: hedef, G: geçici
*/
void hanoiKuleleri (char kaynak, char temp, char hedef, int n)
{
  if (n == 0)
  {
  return;
  }

  // (S,D,T,n-1) (K,H,G,n-1)
  hanoiKuleleri (kaynak, hedef, temp, n-1);

   // move the bottom most disk from 'kaynak' to 'hedef' of the
     current problem
  printf ("\n%d disk (%c -> %c)", n, kaynak, hedef);

  // (T,S,D,n-1)
  hanoiKuleleri (temp, kaynak, hedef, n-1);
}

int main (void)
{
  int n;
```

```
26    printf ("\nNumber of Disks \"n\": ");
      scanf ("%d", &n);

      hanoiKuleleri ('S', 'T', 'D', n);

31    printf ("\n");
      return 0;
  }
```

Eleştiri:

1. Döngü ve özyineleme eyleminde bir blok deyim sürekli tekrar edilir.
2. Her ikisi tekrarı sona erdiren kodlara sahip olmalıdır.
3. Döngü, onu kontrol eden mantıksal ifadenin değerinin olumsuz olmasıyla sonlanır.
4. Yinelemede ise ardışık çağrılar, taban denilen en küçük parçaya erişince sonlanır.
5. Döngü bir yerde sonlanmıyorsa, sonsuz kez aynı işi yapabilir.
6. Benzer şekilde özyinemele eyleminde, çağrılar taban parçaya inemiyorsa, özyineleme sonsuz kez tekrarlanır.
7. Genel olarak döngüde, aynı değişken adresine her adımda bulunan yeni değer yazıldığı için daha az bellek kullanılır.
8. Özyinelemede ise, üstten başlayarak tabana ininceye kadar çağrılan fonksiyon değerleri bellekte tutulur. Dolayısıyla, çağrı sayısı arttıkça, bellek kullanımı artar.
9. Genellikle, özyineleme yöntemiyle çözülen her problem döngü ile de çözülebilir.

Neden Özyineleme Kullanılır? Yukarıdaki eleştiriler ışığıda, şu soru sorulbilir:

> Neden özyinleme yöntemi kullanılıyor?
>
> Bunun iki yanıtı olabilir:

1. Özyineleme yöntemiyle karmaşık problemlerin çözümü, çoğunlukla, döngülerle yapılan çözümlerden daha basit ve daha kısadır. Özyineleme yöntemi, karmaşık problemi *taban* denilen en basit haline indirger, oradan başlayarak adım adım asıl problemi çözer.

2. Bazı dillerde döngü yoktur, onun yerine kendi kendisini çağıran *özyineli* fonksiyonlar kullanılır.

Alıştırmalar 2.1.

1. Aşağıdaki özyineli fonksiyon saklı bir sayıyı bulana kadar kendi kendisini çağırıyor. Fonksiyonu bir program içine yerleştirip koşturunuz.

```
int gizliyiBul(int n)
{
if(n == 0)
return n;
else
return(number + gizliyiBul(n - 1));
}
```

2. Bir sayının sayakları toplamını özyineli fonksiyon ile bulunuz.

Program 2.11.

```
#include<stdio.h>
int main(){
  int n,x;
  clrscr();
  printf("\nBir tam sayı giriniz: ");
  scanf("%d",&n);
  x=sayakTopla(n);
  printf("%d sayısının sayakları toplamı: %d",n,x);
  return 0;
}

int r,s;
int sayakTopla(int n){
if(n){
          r=n%10;
          s=s+r;
          findsum(n/10);
      }
      else
        return s;
}
```

3. Bir sayının istenen bir kuvvetini özyineli fonksiyon ile bulan bir C programı yazınız.

Program 2.12.

```
#include<stdio.h>
int main(){
  int ust,num;
  long int res;
  long int kuvvet(int,int);
```

```
    printf("\nEnter a number: ");
    scanf("%d",&num);
    printf("\nEnter kuvvet: ");
    scanf("%d",&ust);
    res=kuvvet(num,ust);
    printf("\n%d to the kuvvet %d is: %ld",num,ust,res);
    return 0;
}
    int i=1;
    long int sum=1;
    long int kuvvet(int num,int ust){
        if(i<=ust){
            sum=sum*num;
            kuvvet(num,ust-1);
        }
        else
        return sum;
    }
```