BLM 112- Programlama Dilleri II

Hafta 3: İşaretçiler (Kısım 2)

Yrd. Doç. Dr. Caner ÖZCAN

İyilik insanları birbirine bağlayan altın zincirdir.

Goethe

Fonksiyonu Referans ile Çağırma (Call by Reference)

- Bir fonksiyona gönderilen parametrenin normalde değeri değişmez. Fonksiyon içinde yapılan işlemlerin hiçbiri argüman değişkeni etkilemez.
- Sadece değişken değerinin aktarıldığı ve argümanın etkilenmediği bu duruma, "***call by value***" veya "***pass by value***" adı verilir.
- Bir fonksiyondan birden fazla değer döndürülmesi veya fonksiyonun içerisinde yapacağımız değişikliklerin parametre değişkene yansıması gereken durumlar olabilir.
- İşte bu gibi zamanlarda, "***call by reference***" veya "***pass by reference***" olarak isimlendirilen yöntem kullanılır.
- Argüman değer olarak aktarılmaz; argüman olan değişkenin adres bilgisi fonksiyona aktarılır. Bu sayede fonksiyon içerisinde yapacağınız her türlü değişiklik argüman değişkene de yansır.

Fonksiyonu Değer ile Çağırma (Call by Value)

```c
#include <stdio.h>
void arttir(int);
int main14(void)
{
   int i;
   i = 5;
   printf("oncesi %d\n", i);
   arttir(i);
   printf("sonrasi %d\n", i);
   getchar();

   return 0;
}

void arttir(int k)
{
   k++;
}
```

Fonksiyonu Referans ile Çağırma (Call by Reference)

```c
#include <stdio.h>
void increment(int *);
int main(void)
{
   int i;
   i = 5;
   printf("oncesi %d\n", i);
   increment(&i);
   printf("sonrasi %d\n", i);
   getchar();

   return 0;
}

void increment(int *k)
{
   (*k)++;
}
```

Fonksiyonu Referans ile Çağırma (Call by Reference)

- Eğer yazdığınız fonksiyon birden çok değer döndürmek zorundaysa, referans yoluyla aktarım zorunlu hâle geliyor.
- Çünkü return ifadesiyle sadece tek bir değer döndürebiliriz.
- Örneğin bir bölme işlemi yapıp, bölüm sonucunu ve kalanı söyleyen bir fonksiyon yazacağımızı düşünelim.
- Bu durumda, bölünen ve bölen fonksiyona gidecek argümanlar olurken; kalan ve bölüm geriye dönmelidir.
- return ifadesi geriye tek bir değer vereceğinden, ikinci değeri alabilmek için referans yöntemi kullanmamız gerekir.

Fonksiyonu Referans ile Çağırma (Call by Reference)

```c
#include<stdio.h>
int bolme_islemi( int, int, int * );
int main( void )
{
    int bolunen, bolen;
    int bolum, kalan;
    bolunen = 13;
    bolen = 4;
    bolum = bolme_islemi( bolunen, bolen, &kalan );
    printf( "Bolum: %d Kalan: %d\n", bolum, kalan );
    getchar();
    return 0;
}
int bolme_islemi( int bolunen, int bolen, int *kalan )
{
    *kalan = bolunen % bolen;
    return bolunen / bolen;
}
```

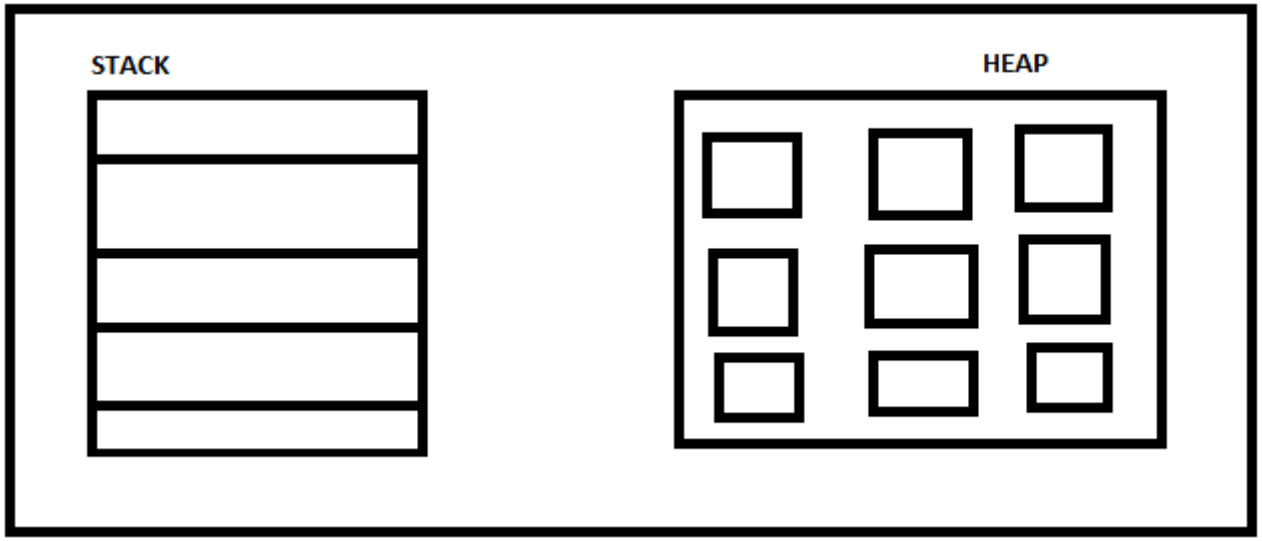
Dinamik Bellek Yönetimi

- Bir program çalıştırıldığında işletim sistemi programın çalışması için bir alan ayırır (stack ve heap).
- Stack, global –statik değişkenler, fonksiyonlar ve onların yerel değişkenlerinin tutulduğu yerdir.
- Heap, program için ayrılan ve çalışma zamanında hafıza alanı açmak için kullanılan boş alandır.

Stack (Yığın) ve Heap (Öbek)

- Yığın ve öbek belleğin mantıksal parçalarıdır.
- Yığın LIFO (Last In First Out) mantığında çalışır. Bir kutu olarak düşündüğünüzde kutuya koyduğunuz kitaplar üst üste eklenirken, son koyduğunuz kitaba ilk erişirsiniz.
- Heap ise program için ayrılmış olan ve programcının sorumluluğunda kullanılan bellek bölümüdür.

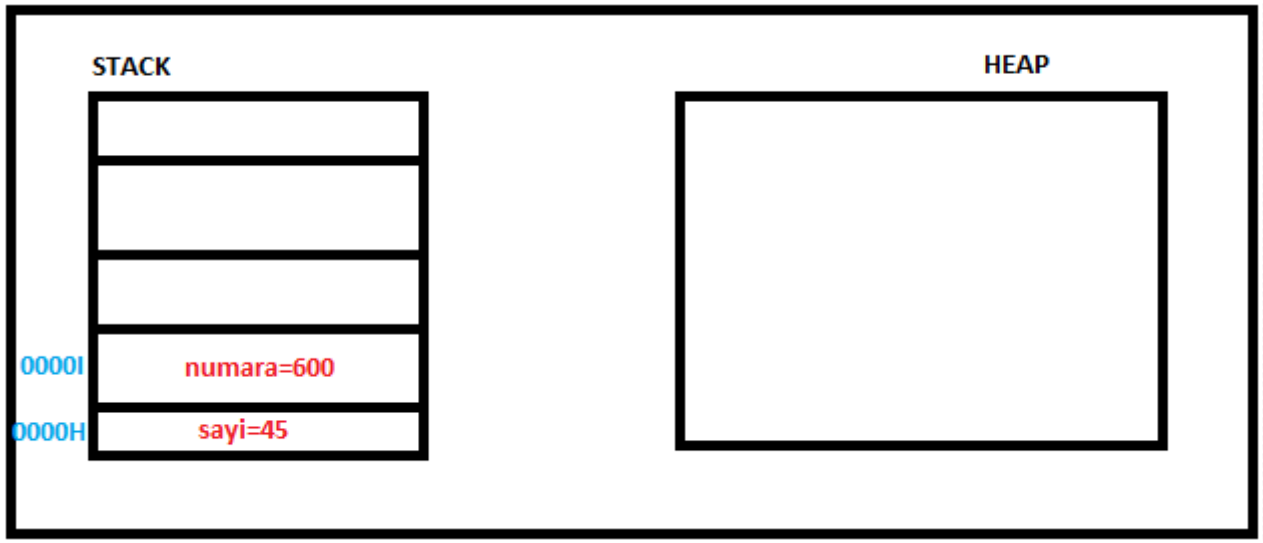
Stack (Yığın) ve Heap (Öbek)



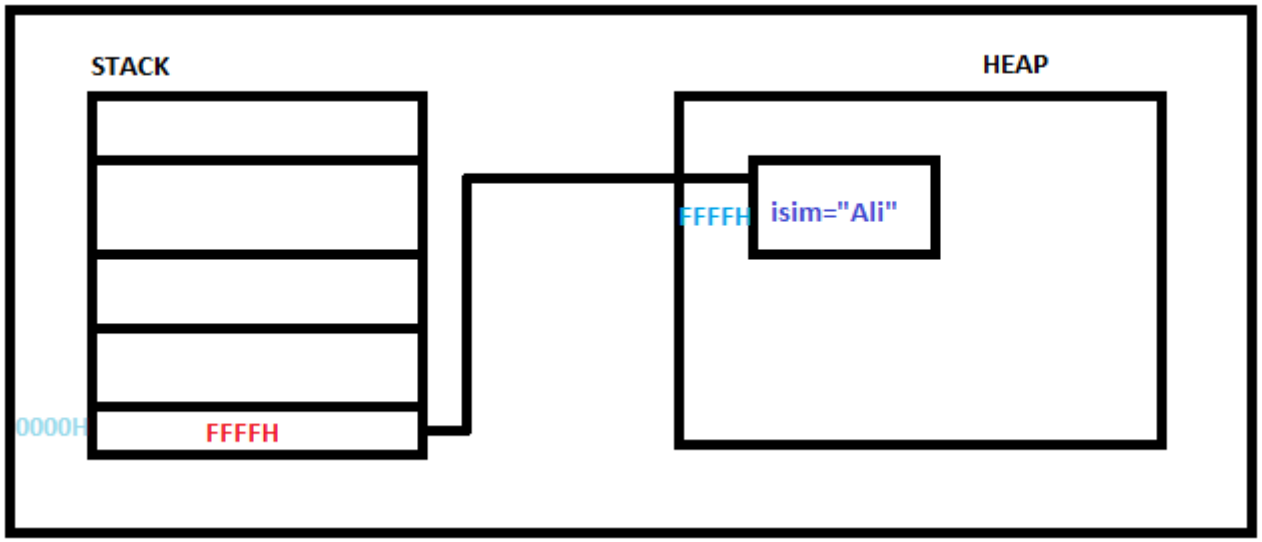
Stack (Yığın) ve Heap (Öbek)

- Yığın içerisinde değer tipi değişkenleri, işaretçi değişkenleri ve kod adreslerini saklarken, işaretçiler tarafından gösterilen bellek alanları ise öbek alanında saklanır.
- Yığın daha hızlıdır. Çalışma mantığı basittir ve elemanlar sıralı biçime yerleştirilmiştir.
- Öbek daha yavaştır. Öbekteki nesneye erişmek için karmaşık arama işlemi yapılmalıdır.

Stack (Yığın) ve Heap (Öbek)



Stack (Yığın) ve Heap (Öbek)



Dinamik Bellek Yönetimi

- Şimdiye kadar yazdığımız programlarda örneğin dizi oluştururken dizinin boyutu önceden belliydi.
- Ancak dizi elemanı sayısı sabit olmayan ve önceden kestiremediğimiz bir durum varsa ve biz ne olur ne olmaz diyerek dizi eleman sayısı olarak büyük bir sayı atarsa bu seferde hafızayı boş yere işgal etmiş olacağız.
- Çözüm dinamik bellek yönetimi kullanmaktır.

Dinamik Bellek Yönetimi

- Dinamik bellek yönetiminde program akışında ihtiyaç duyulan bellek miktarı belirlenir.
- Dinamik bellek yönetimi için ***calloc(), malloc()*** ve ***realloc()*** olmak üzere üç fonksiyon kullanılır.
- Her üç fonksiyon da ***stdlib*** kütüphanesinde bulunur. Bu yüzden fonksiyonlardan herhangi birini kullanacağınız zaman, programın başına ***#include<stdlib.h>*** yazılması gerekir.

malloc

- Malloc fonksiyonu bir değişken için hafızadan bir blok yer ayrılması için kullanılır.
- Eğer hafızada yeterli alan yoksa fonksiyon NULL döndürür.

```
int *ptr;
ptr = (int *) malloc(n*sizeof(int));
```

calloc

- Calloc fonksiyonu da hafıza bloğu almak için kullanılabilir.
- Eğer hafızada yeterli alan yoksa fonksiyon NULL döndürür.
- Malloc fonksiyonundan farklı olarak argüman listesi değişmektedir.

```
char *ptr;
ptr = (char *)calloc(10, sizeof(char));
```

realloc

- realloc fonksiyonu hafızadan ayrılan bir alanı yeniden boyutlandırmak için kullanılır.
- Tekrar ayarlanacak hafıza alanının başlangıcını işaret edecek bir pointer ve yeni boyut bilgisini parametre olarak alır.

```
void *realloc(void *ptr, size_t size);
```

free

- Gelişmiş programlama dillerinde (örneğin, Java, C#, vb...) kullanılmayan nesnelerin temizlenmesi otomatik olarak çöp toplayıcılarla (Garbage Collector) yapılmaktadır.
- Ne yazık ki C programlama dili için bir çöp toplayıcı yoktur ve iyi programcıyla, kötü programcı burada kendisini belli eder.

free

- Büyük boyutta ve kapsamlı bir program söz konusuysa, efektif bellek yönetiminin ne kadar önemli olduğunu daha iyi anlarsınız.
- Gereksiz tüketilen bellekten kaçınmak gerekmektedir.
- Alloc ve malloc fonksiyonlarıyla hafızadan ayrılan alanın tekrar heap alanına serbest bırakılması için kullanılır.

```
int *ptr;
ptr = (int *) malloc(n*sizeof(int));
free(ptr);
```

Örnek-1

```c
#include <stdio.h>
#include <stdlib.h>
int main(void)
{
    int n,i,*ptr,sum=0;
    printf("Eleman sayısını girin\n");
    scanf("%d",&n);

    ptr= (int *)malloc(n*sizeof(int));
    if(ptr==NULL)
    {
        printf("Yeterli hafıza yok");
    }
    printf("Dizi elemanlarını girin\n");
    for(i=0;i<n;i++)
    {
        scanf("%d",ptr+i);
        sum += *(ptr+i);
    }
    printf("Toplam = %d",sum);
    getchar();
    getchar();
    return 0;
}
```

Örnek-2

```c
#include <stdio.h>
#include<stdlib.h>
int *dizileri_birlestir( int [], int, int [], int );
int main( void )
{
    int i;
    int liste_1[5] = { 6, 7, 8, 9, 10 };
    int liste_2[7] = {13, 7, 12, 9, 7, 1, 14 };
    // sonucun dondurulmesi icin pointer tanimliyoruz
    int *ptr;

    ptr = dizileri_birlestir( liste_1, 5, liste_2, 7 );

    // ptr isimli pointer'i bir dizi olarak dusunebiliriz
    for( i = 0; i < 12; i++ )
        printf("%d ", ptr[i] );
    printf("\n");

    return 0;
}
```

Örnek-2

```
21  int *dizileri_birlestir( int dizi_1[], int boyut_1,
22                           int dizi_2[], int boyut_2 )
23  {
24      int *sonuc = (int *)calloc( boyut_1+boyut_2, sizeof(int) );
25      int i, k;
26      // Birinci dizinin degerleri ataniyor.
27      for( i = 0; i < boyut_1; i++ )
28          sonuc[i] = dizi_1[i];
29
30      // Ikinci dizinin degerleri ataniyor.
31      for( k = 0; k < boyut_2; i++, k++ ) {
32          sonuc[i] = dizi_2[k];
33      }
34
35      // Geriye sonuc dizisi gonderiliyor.
36      return sonuc;
37  }
```