BLM-112 PROGRAMLAMA DİLLERİ II

Ders-3 İşaretçiler (Pointer) (Kısım-2)

Yrd. Doç. Dr. Ümit ATİLA
umitatila@karabuk.edu.tr
http://web.karabuk.edu.tr/umitatilla/

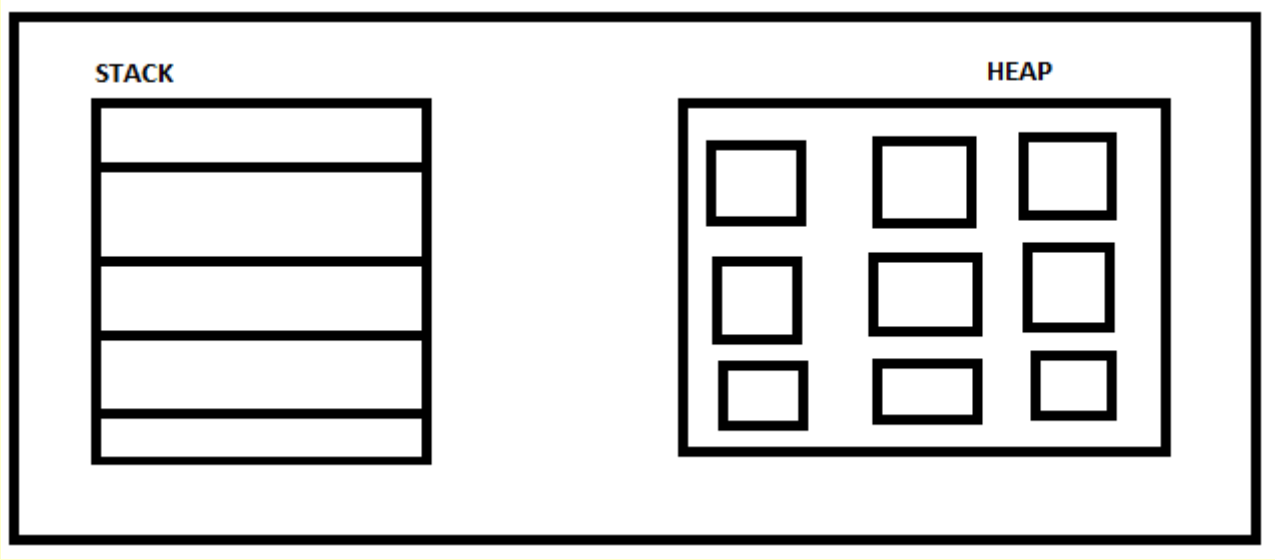
Dinamik Bellek Yönetimi

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

Stack ve Heap

- Stack ve heap hafızanın mantıksal bölümleridir.
- Stack LIFO (Son giren ilk çıkar) mantığıyla çalışır. Bir kutu gibi düşünülebilir.
- Kutuya bir şeyler koyarsınız ve son koyduğunuz nesneyi ilk önce alırsınız.
- Heap ise programcı için bir tarla gibidir ve kullanımı programcının sorumluluğundadır.

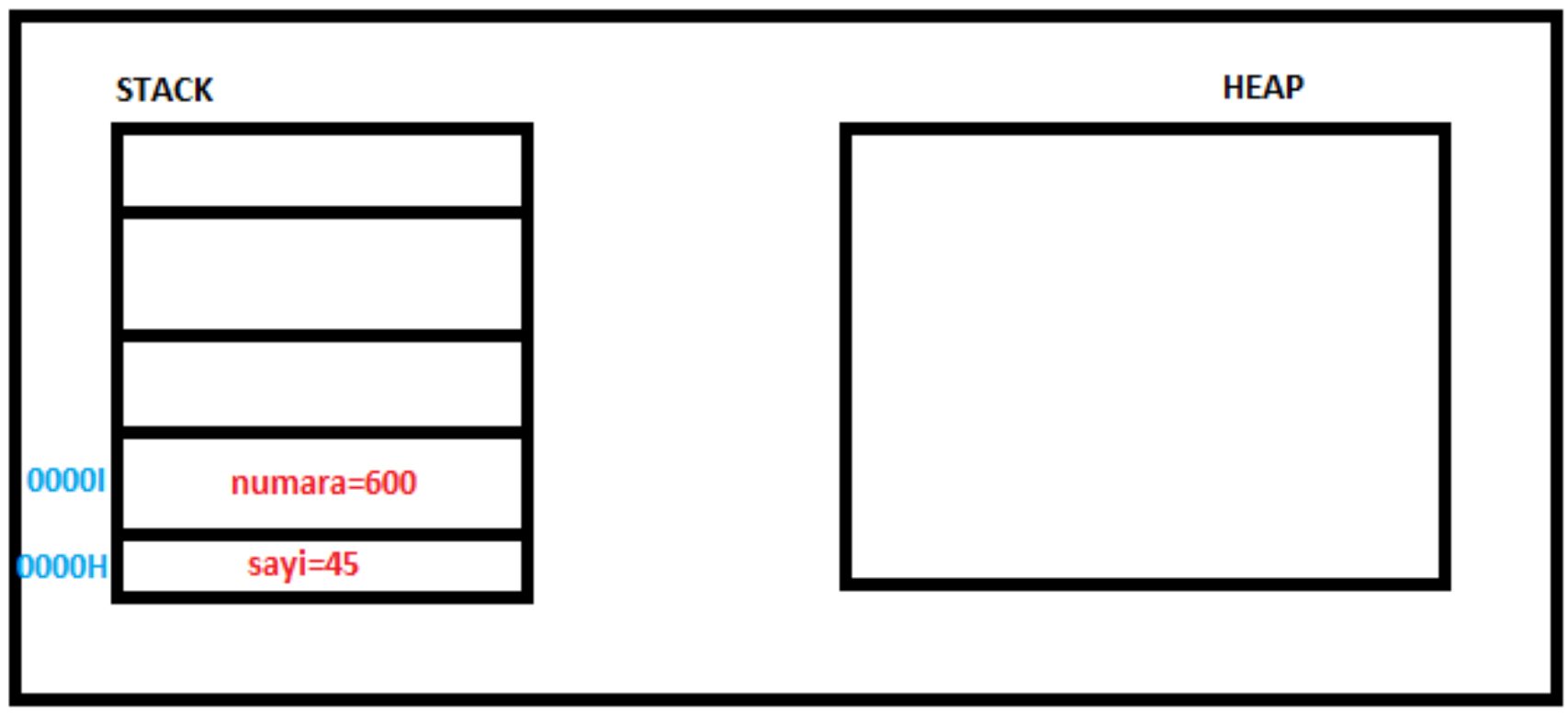
Stack ve Heap



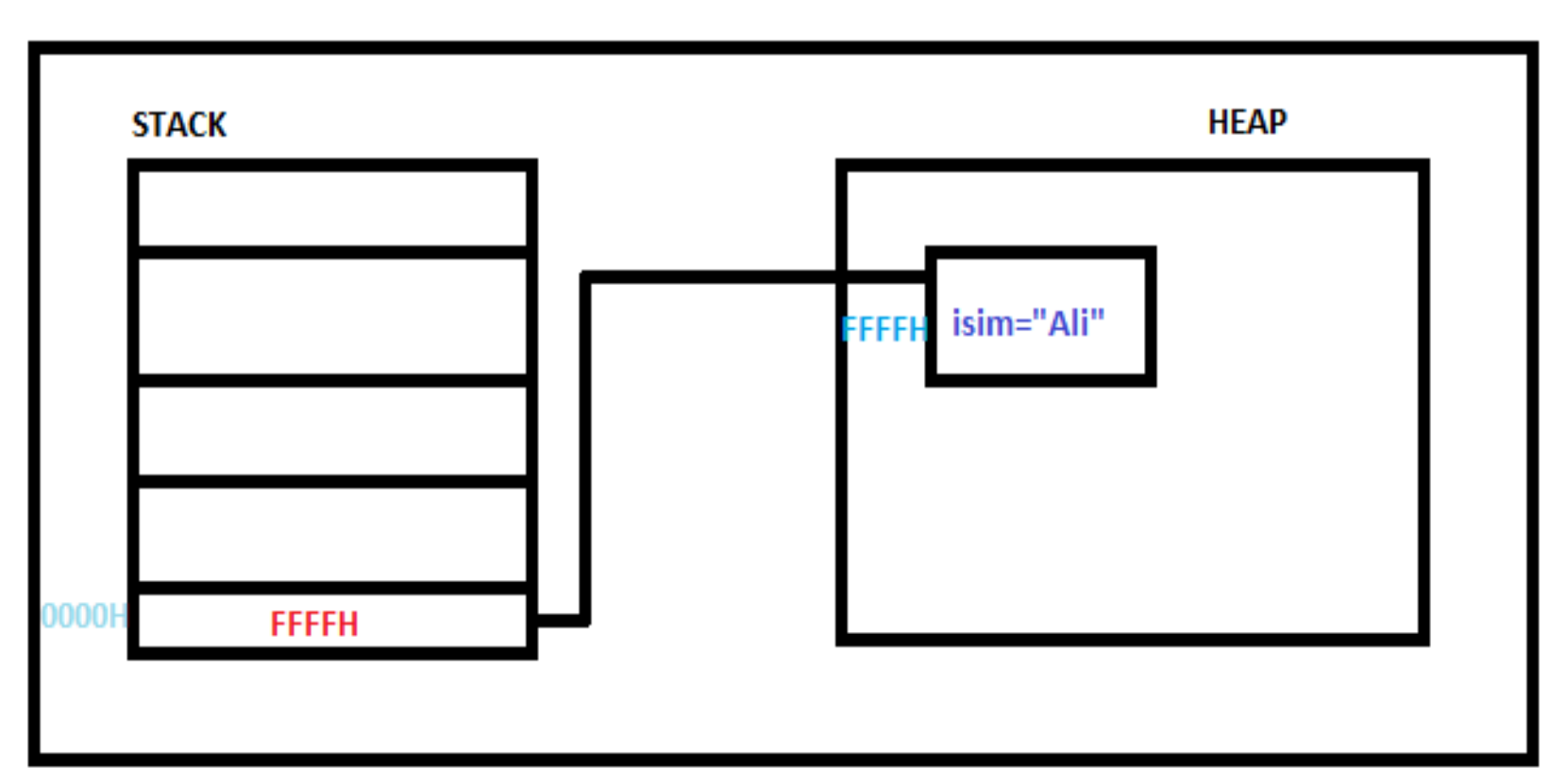
Stack ve Heap

- Değişkenler, işaretçiler ve kod adresleri stack bölgesinde tutulurken, işaretçiler tarafından gösterilen hafıza alanları ise heap alanında tutulur.
- Stack heap alanına göre daha hızlıdır. Çünkü stack bölgesinin çalışma mantığı kolaydır. Erişmek istediğimiz alanlar bir biri ardına sıralanmıştır.
- Heap erişimi ise daha yavaştır. Çünkü heap alanında boş bir bölgeye yerleştirdiğimiz bir nesneye erişmek için heap alanında karmaşık bir arama yapmalıyız.

Stack ve Heap



Stack ve Heap



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 atarsak bu seferde hafızayı boş yere işgal etmiş olacağız.
- Çözüm dinamik bellek yönetimi kullanmaktır.
- Dinamik hafıza yönteminde ihtiyaç duyulan hafıza miktarı programın çalışması esnasında belirlenir.

Dinamik Bellek Yönetimi

- 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 ilk değer ataması yapar

```
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);
```

- Realloc fonksiyonu daha önce tahsis edilen bloğun hemen altında sürekliliği bozmayacak şekilde tahsisat yapar.
- Eğer daha önce tahsis edilen bloğun altında yeterli alan yoksa bellekte bloğun tamamı için yer arar.
- Yeterli hafıza alanı bulursa bloğun tamamını hafızada o bölgeye taşır. Eskisini siler. Yeterli alan bulamaz ise NULL döndürür.

realloc

- Eğer realloc fonksiyonu yeterli yer bulamayıp bloğu başka yere taşırsa bu durumda geri dönüş değerini aynı işaretçiye atamak gerekir.
- Çünkü bu durumda bloğun başlangıç adresi değişmektedir.

```
char *ptr;
ptr = (char *)calloc(10, sizeof(char));
ptr = realloc (ptr, 20*sizeof(char));
```

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 her hafızadan ayrılan alanın tekrar heap alanına serbest bırakılması için free fonksiyonu çağırılır.

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

Örnek-1

```c
#include <stdio.h>
#include <stdlib.h>

void main()
{
    int *p, n, i;
    printf("Eleman sayisi gir:");
    scanf("%d", &n);

    p = (int *) malloc(n*sizeof(int));
    if(p == NULL)
        printf("Yeterli hafiza yok");
    else
    {
        for(i=0;i<n;i++){
            printf("Sayi gir:");
            scanf("%d", p+i);
        }
    }
    printf("Adres \t\t Deger\n");
    for(i=0;i<n;i++)
        printf("%d \t\t %d\n", p+i, *(p+i));

    free(p);
}
```

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  }
```

Örnek-3

```
void main()
{
    int *p, n, i;
    printf("Eleman sayisi gir:");
    scanf("%d", &n);

    p = (int *) malloc(n*sizeof(int));
    if(p == NULL)
        printf("Yeterli hafiza yok");
    else
    {
        for(i=0;i<n;i++){
            printf("Sayi gir:");
            scanf("%d", p+i);
        }
    }
    printf("Adres \t\t Deger\n");
    for(i=0;i<n;i++)
        printf("%d \t\t %d\n", p+i, *(p+i));

    p = (int *) realloc(p, (n+3)*sizeof(int));

    printf("Adres \t\t Deger\n");
    for(i=0;i<n+3;i++)
        printf("%d \t\t %d\n", p+i, *(p+i));
}
```

Kaynaklar

- Paul J. Deitel, "C How to Program", Harvey Deitel.
- Kaan Aslan, "A'dan Z'ye C Klavuzu 8. Basım", Pusula Yayıncılık, 2002.