

CME 112- Programming Languages II

Week 5 Pointers

Assist. Prof. Dr. Caner Özcan

People never make mistakes are people who do nothing. And the biggest mistake in life is to think yourself perfect.
~Y. Emre

Call by Value and Call by Reference

- ▶ Normally a value of parameter sent to a function does not change. And modifications in function does not effect original variable.
- ▶ The case in which the original variable is not changed but its copy is sent to a function is called "***call by value***" or "***pass by value***".
- ▶ Sometimes we need to return more than one value from a function or we need the original variable changed by the function.

Call by Value and Call by Reference)

- ▶ For this purposes we use "*call by reference*" or "*pass by reference*"
- ▶ In call by reference, arguments are not passed with their values, but with their addresses. Thus, all modifications on arguments effect the original variable.



Call by Value

```
1  #include <stdio.h>
2  void arttir(int);
3  int main14(void)
4  {
5      int i;
6      i = 5;
7      printf("oncesi %d\n", i);
8      arttir(i);
9      printf("sonrasi %d\n", i);
10     getchar();
11
12     return 0;
13 }
14
15 void arttir(int k)
16 {
17     k++;
18 }
```

Call by Reference

```
1  #include <stdio.h>
2  void increment(int *);
3  int main(void)
4  {
5      int i;
6      i = 5;
7      printf("oncesi %d\n", i);
8      increment(&i);
9      printf("sonrasi %d\n", i);
10     getchar();
11
12     return 0;
13 }
14
15 void increment(int *k)
16 {
17     (*k)++;
18 }
```

Call by Reference

- ▶ If your function has to return more than one value, pass by reference usage is necessary.
- ▶ Because return keyword can only return one value from function.
- ▶ For example, we want to write a division function that gives division result and remainder.
- ▶ In this case, divided number and divisor is sent to function and remainder and division should be returned back from function.
- ▶ As return keyword can only return one value, second value must be returned by reference method.

Call by Reference

```
1  #include<stdio.h>
2  int bolme_islemi( int, int, int * );
3  int main( void )
4  {
5      int bolunen, bolen;
6      int bolum, kalan;
7      bolunen = 13;
8      bolen = 4;
9      bolum = bolme_islemi( bolunen, bolen, &kalan );
10     printf( "Bolum: %d Kalan: %d\n", bolum, kalan );
11     getchar();
12     return 0;
13 }
14 int bolme_islemi( int bolunen, int bolen, int *kalan )
15 {
16     *kalan = bolunen % bolen;
17     return bolunen / bolen;
18 }
```

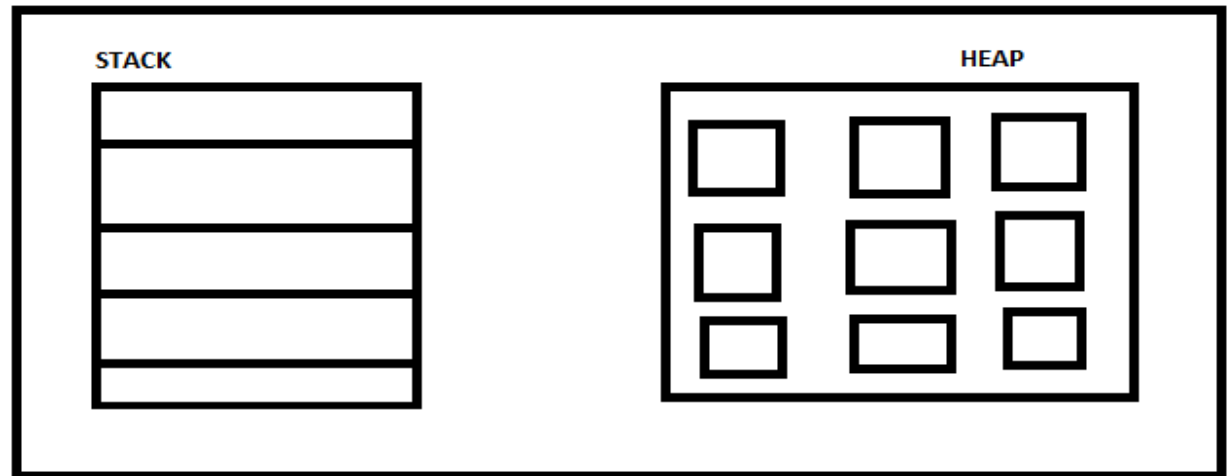
Dynamic Memory Allocation

- ▶ When a program executes, the operating system reserves space to run program (stack and heap).
- ▶ The stack is memory space where functions and their locally defined variables reside.
- ▶ The heap is reserved for program and it is an empty section to use for allocating memory at runtime.



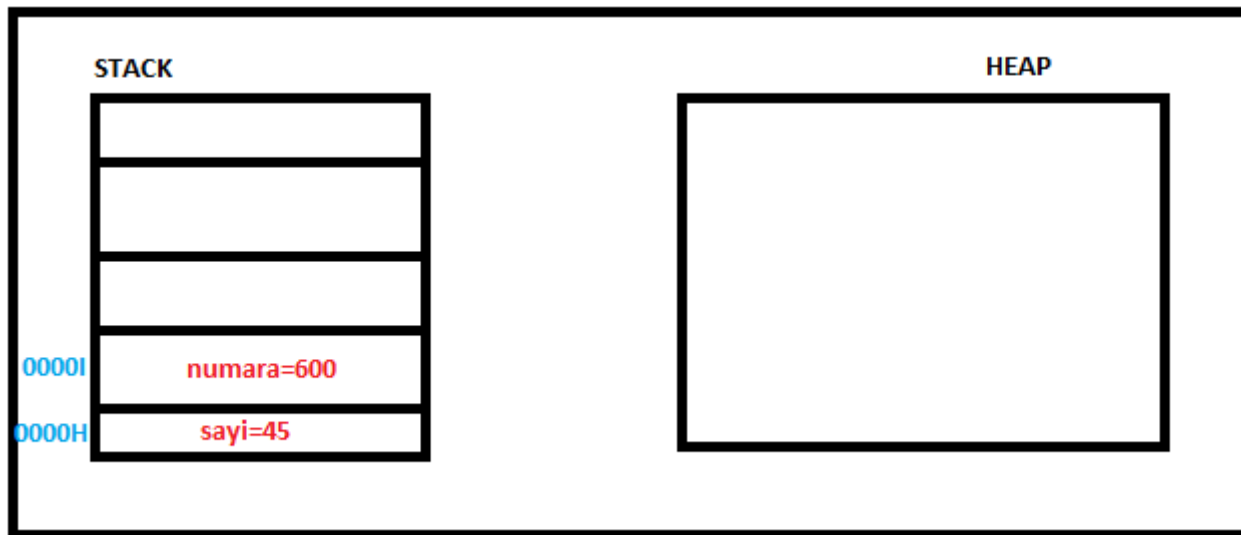
Stack and Heap

- ▶ Stack and heap are the logical parts of memory.
- ▶ Stack works in LIFO (Last in First Out) principal. If considered as a box: one of the books that you put in the box is placed on top of the other. Latest added book is accessed first.
- ▶ Heap is like a farm of programmer and usage of it is in responsibility of programmer.



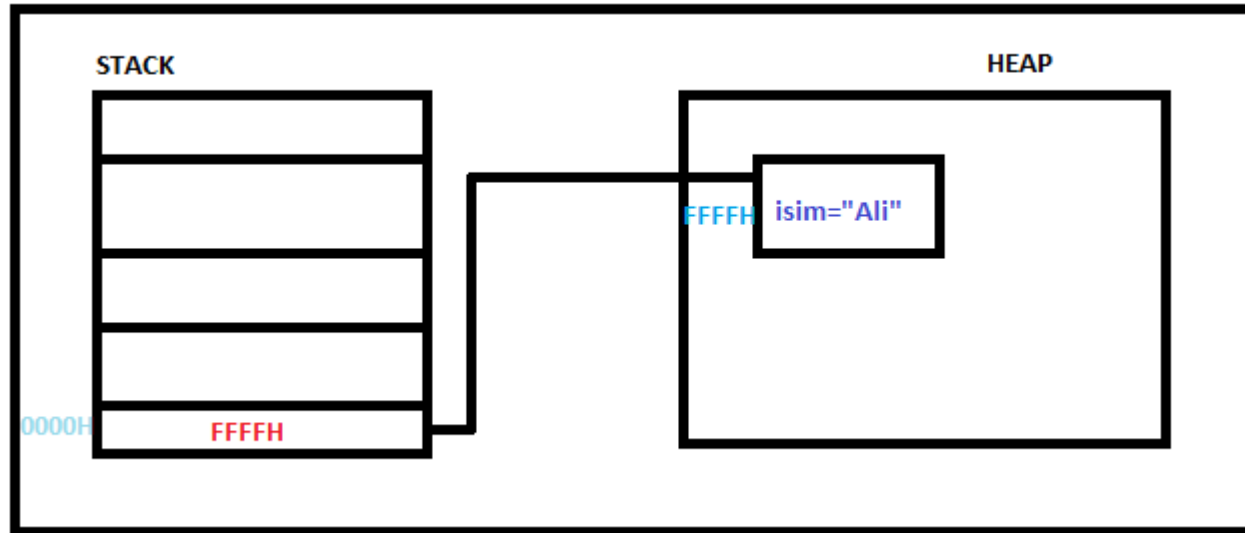
Stack and Heap

- ▶ While we store value type variables, pointer variables and code addresses in the stack.
- ▶ Stack is faster than heap. Because working principal of stack is easy and spaces that we want to reach are placed one after the other.



Stack and Heap

- ▶ Memory spaces that is shown by pointers are stored in the heap space.
- ▶ Heap is slower than stack. Because to reach an object in the heap we should perform a complex search as we put an object into any empty space in heap.



Dynamic Memory Allocation

- ▶ We may need an array whose number of elements may vary according to needs.
- ▶ For such kind of need, creating a large array to solve the problem may consume memory in vain.
- ▶ More effective solution is usage of dynamic memory allocation.



Dynamic Memory Allocation

- ▶ In dynamic memory allocation, amount of memory needed is determined during the execution of program.
- ▶ *malloc*, *calloc*, or *realloc* are the three functions used to manipulate memory.
- ▶ These commonly used functions are available through the **stdlib** library, so you must include this library in order to use them.

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

A decorative horizontal bar at the bottom of the slide, composed of several colored rectangular segments in shades of blue, green, orange, red, and grey.

• Malloc() Function

- ▶ Malloc function is used to allocate a block of memory for one variable.
- ▶ If there is not enough memory available, malloc will return NULL.

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



Calloc() Function

- ▶ Calloc function is also used to allocate a block of memory.
- ▶ If there is not enough memory available, calloc will return NULL.
- ▶ Unlike malloc function, it takes two arguments.

```
char *ptr;
```

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

Realloc() Function

- ▶ Realloc is used to resize an allocated memory space.
- ▶ A pointer that will point the starting address of resized memory space and new size are passed to realloc function as parameter.

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



Free() Function

- ▶ In high level programming languages such as (C#, Java) removing unused objects from memory is achieved automatically by Garbage Collector.
- ▶ Unfortunately, there is no garbage collector for C language and bad and good programmer is separated easily with this issue.

Free() Function

- ▶ How important an effective memory management is may be understood when we write large programs.
- ▶ We should avoid consuming unnecessary memory.
- ▶ Every call to an malloc or calloc function you must have a corresponding call to free.

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



Example 1

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 int main(void)
4 {
5     int n,i,*ptr,sum=0;
6     printf("Eleman sayısını girin\n");
7     scanf("%d",&n);
8
9     ptr= (int *)malloc(n*sizeof(int));
10    if(ptr==NULL)
11    {
12        printf("Yeterli hafıza yok");
13    }
14    printf("Dizi elemanlarını girin\n");
15    for(i=0;i<n;i++)
16    {
17        scanf("%d",ptr+i);
18        sum += *(ptr+i);
19    }
20    printf("Toplam = %d",sum);
21    getchar();
22    getchar();
23    return 0;
24 }
```

Example 2

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 int *dizileri_birlestir( int [], int, int [], int );
4 int main( void )
5 {
6     int i;
7     int liste_1[5] = { 6, 7, 8, 9, 10 };
8     int liste_2[7] = {13, 7, 12, 9, 7, 1, 14 };
9     // sonucun dondurulmesi icin pointer tanimliyoruz
10    int *ptr;
11
12    ptr = dizileri_birlestir( liste_1, 5, liste_2, 7 );
13
14    // ptr isimli pointer'i bir dizi olarak dusunebiliriz
15    for( i = 0; i < 12; i++ )
16        printf("%d ", ptr[i] );
17    printf("\n");
18
19    return 0;
20 }
```

Example 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 }
```

Next Week

- ▶ Examples with Pointers



References

- ▶ Doç. Dr. Fahri Vatansever, “Algoritma Geliştirme ve Programlamaya Giriş”, Seçkin Yayıncılık, 12. Baskı, 2015.
- ▶ Kaan Aslan, “A’dan Z’ye C Klavuzu 8. Basım”, Pusula Yayıncılık, 2002.
- ▶ Paul J. Deitel, “C How to Program”, Harvey Deitel.
- ▶ “A book on C”, All Kelley, İra Pohl

Q u e s t i o n s
A n y
?



Thanks for listening

CANER ÖZCAN

 canerozcan@karabuk.edu.tr

Foot note..

The first step is to establish that something is possible; then probability will occur.

