CME 112- Programming Languages II
Lecture 3: Pointers (Part 2)

Assist. Prof. Dr. Ümit ATİLA

Dynamic Memory Alocation

- When a program executes, the operating system gives it a stack and a heap to work with.
- The stack is where global variables, static variables, and functions and their locally defined variables reside.
- The heap is a free section for the program to use for allocating memory at runtime.

Dynamic Memory Alocation

- 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 using dynamic memory allocation.

Dynamic Memory Alocation

- 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>

malloc

- Use the malloc function to allocate a block of memory for a variable.
- If there is not enough memory available, malloc will return NULL.

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

calloc

- You can also ask for multiple blocks of memory with the calloc function.
- If there is not enough memory available, calloc will return NULL.
- Unlike malloc function, performs an initial value assignment.

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

realloc

- 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

- 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

- How important an effective memory management is may be understood when we write large programs.
- We should avoid consuming memory in vain.
- 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);
```

Sample-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;
}
```

Sample-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;
}
```

Sample-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  }
```

Pointers & Structs

- Structs may be passed to functions with a pointer

```
struct ogrenci{
        char no[10];
        int  notu;
};
struct ogrenci *a;
```

- For accessing the space allocated for variable a:

```
*a.notu=56;
strcpy((*a).no, "95001");
```

- An other way of this

```
a->notu=56;
strcpy(a->no, "95001");
```

Pointers & Structs

```c
#include <stdio.h>
typedef struct {
     char adi[35];
     char adres1[40];
     char adres2[40];
     char tel[15];
     float borc;
} kisiler;
void yaz(kisiler *z);
int main()
{
   kisiler  a;

   printf("Adını gir   : "); gets(a.adi);
   printf("Adres-1     : "); gets(a.adres1);
   printf("Adres-2     : "); gets(a.adres2);
   printf("Telefonu    : "); gets(a.tel);
   printf("Borcu       : "); scanf("%f", &(a.borc));
   yaz(&a);
   getchar();
   getchar();
   return 0;
}
void yaz(kisiler *z)
{

  printf("Adı          : "); puts(z->adi);
  printf("Adresi       : "); puts(z->adres1);
  printf("             : "); puts(z->adres2);
  printf("Telefonu     : "); puts(z->tel);
  printf("Borcu        : "); printf("%.0f\n", z->borc);
}
```

Dynamic Memory Allocation & Arrays

```c
#include<stdio.h>
#include<stdlib.h>
int main( void )
{
    // Dinamik bir dizi oluşturmak icin pointer kullaniriz.
    int *dizi;
    // Dizimizin kac elemanli olacagini eleman_sayisi isimli degiskende tutuyoruz.
    int eleman_sayisi;
    int i;
    printf( "Eleman sayısını giriniz> ");
    scanf( "%d", &eleman_sayisi );
    // malloc( ) fonksiyonuyla dinamik olarak dizimizi istedigimiz boyutta oluşturalım.
    dizi = (int *)malloc( eleman_sayisi * sizeof( int ) );
    //dizi = (int *)calloc( eleman_sayisi, sizeof( int ) );

    for( i = 0; i < eleman_sayisi; i++ )
        printf( "Adres:%d\tDeger:%d\n", &dizi[i],dizi[i] );

    // hafizadan temizleme
    free( dizi );

    while( getchar() != '\n' ) { /*do nothing*/} ;
        getchar() ; /* wait */
    return 0;
}
```

Function Pointers

- A pointer to a function contains the address of the function in memory.
- A function name is really the starting address in memory of the code that performs the function's task.
- int (*fPtr) (int,int)
  - In this definition, fPtr shows the adress of a function that takes two integer parameters and returns an integer value.
- int *fPtr (int,int)
  - In this definition, a function named fPtr is defined that takes two integer parameters and returns an integer pointer.

Function Pointers

```c
#include <stdio.h>
int kare(int);
int kup(int);
int main(void)
{
    /* bir int değer alıp geriye int değer gönderen bir fonksiyonun adresi  */
    int (*islem)(int);
    int i;
    char c;

    printf("1-kare alani\n2-kup hacmi\n ");
    c = getchar();
    printf("\nSayıyı gir : ");
    scanf("%d", &i);
    if (c == '1')
        islem = kare;    /* kare işlevinin adresi islem değişkenine kopyalanır */
    else
        islem = kup;
    printf("Sonuç = %d\n", islem(i));
    while( getchar() != '\n' ) { /*do nothing*/} ;
        getchar() ; /* wait */
}
int kare(int s)
{
    return s*s;
}
int kup(int s)
{
    return s*s*s;
}
```

Void Pointers

- Pointers can be defined in void type.
- We have to specify the type of data for accessing the data that void pointer show.

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
int main(void)
{
  void *a;
  a = (char*) malloc(6);
  strcpy((char *)a,"12345");
  printf("%s\n",  a);
  free(a);
  a = (double*) malloc(sizeof(double));
  /* değere erişirken veri tipi belirt */
  *(double*)a = 3.123;
  printf("%f\n", *(double *)a);
  getchar();
}
```