CME 112- Programming Languages II
Lecture 2: Pointers (Part 1)

Assist.Prof. Dr. Ümit ATİLA

Memory Structure

- When a variable defined it is stored somewhere in memory
- Memory can be thought as block consist of cells.
- When a variable defined, required number of cell from memory is allocated for the variable.
- How many cell will be reserved for the variable depends on the type of variable.

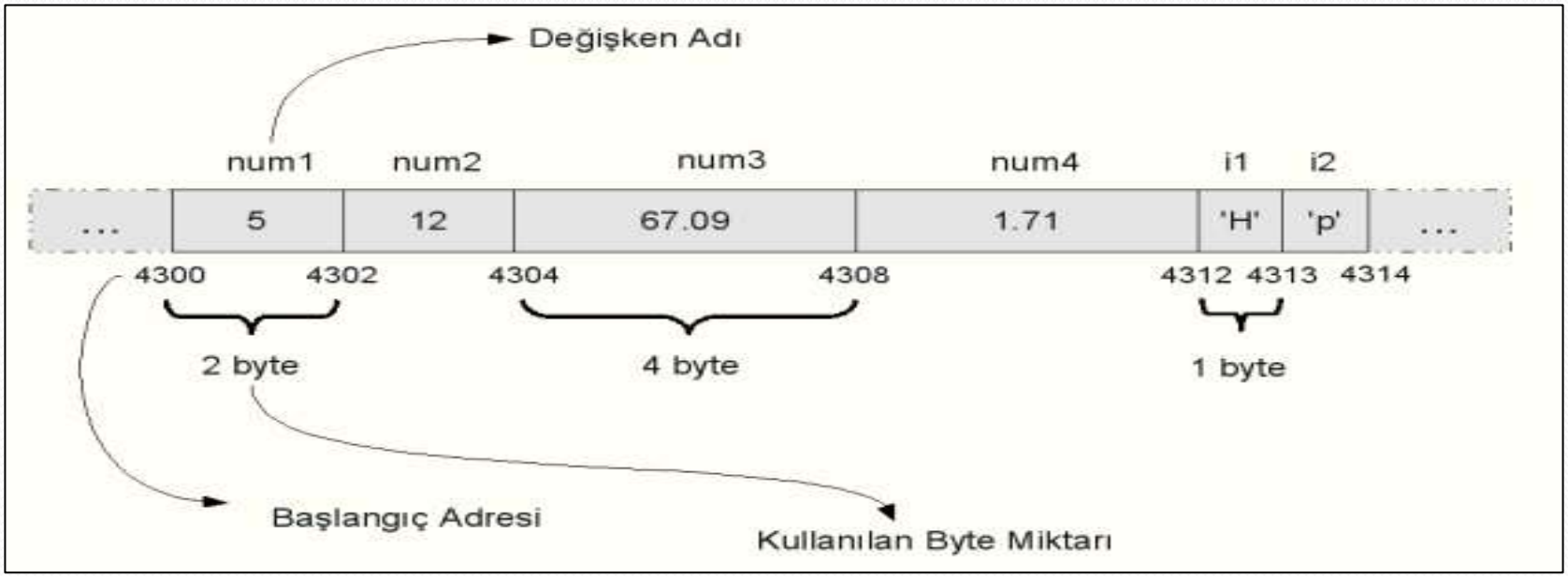
Memory Structure

```c
#include <stdio.h>

int main(void)
{
    // Degiskenler tanımlanıyor:
    int num1, num2;
    float num3, num4;
    char i1, i2;
    // Degiskenlere atama yapiliyor:
    num1 = 5;
    num2 = 12;
    num3 = 67.09;
    num4 = 1.71;
    i1 = 'H';
    i2 = 'p';
    return 0;
}
```

Memory Structure

- If we illustrate the structure of memory after the code in previous slide.
  - Assume that size of int is 2 byte, size of float 4 byte, size of char byte
  - Each cell represents 1 byte space
  - Memory portion for defined variables starts from the adress 4300



Memory Structure

- When a variable is defined, a space required for the variable is reserved in the memory
- E.g.definition ***int num1*** reserves 2 byte space for variable num1
- After that if the value 5 is assigned on variable num1, 5 is stored in memory location allocated for that variable.
- Actually, all operations taken on variable num1 is the modification of cells in the memory location between 4300 and 4302.
- Variable is actually a memory location reserved for a particular label.

Defining Pointers

- Pointer is a data type that shows the memory address of a data block.

  veri tipi *p;

- Variable p stores the **address** of a variable which is in <veri tipi> type

  int *iptr;

  float *fptr;

- The only thing we should pay attention is defining pointer suitable for the data type it points.
- A float variable must only be pointed by a float type pointer.

Defining Pointers

- To make a pointer show the address of a variable, address of the variable should be assigned to pointer.
- For this purpose we should know the adress of the memory location used for the variable.
- It is possible with address operator &..
  - &y → gives the address of variable y.

```
int y = 5;
int *yPtr;
yPtr = &y;
```

Defining Pointers

- After assigning the address of a variable to a pointer. Pointer starts to show the address of related variable.
- If we want to access or modify the value of a variable with pointer, we should use * character in the begining of pointer name.
- All modifications done with * character in the begining of pointer name effects the original variable.

Defining Pointers

```c
#include <stdio.h>

int main(void)
{
   int i;
   int *iptr;
   i = 5;
   iptr = &i;

   printf("i adresi    %p\n", &i);
   printf("iptr degeri %p\n", iptr);

   printf("i degeri    %d\n", i);
   printf("*iptr degeri %d\n", *iptr);

   getchar();
   return 0;
}
```

Defining Pointers (Accessing Variables by Pointers)

- For accessing the value of a variable with pointer, we should use * character in the begining of pointer name

```c
#include<stdio.h>
int main()
{
    int i;
    int *iptr;
    iptr = &i;
    *iptr = 8;
    printf("i değişkeninin değeri %d\n", i);
    printf("iptr adresinin içeriği %d\n", *iptr);

    getchar();
    return 0;
}
```

Defining Pointers (Associating Variables with Pointers)

```c
#include<stdio.h>
int main( void )
{
    // int tipinde değişken tanımlıyoruz:
    int xyz = 10, k;
    // int tipinde pointer tanımlıyoruz:
    int *p;

    // xyz değişkeninin adresini pointer'a atıyoruz.
    // Bir değişken adresini '&' işaretiyle alırız.
    p = &xyz;

    // k değişkenine xyz'nin değeri atanır. Pointer'lar değer tutmaz.
    // değer tutan değişkenleri işaret eder.
    //Başına '*' koyulduğunda, işaret ettiği değişkenin değerini gösterir.
    k = *p;

    return 0;
}
```

Defining Pointers

- We can change the variable that pointer shows during our program continuously.

```c
#include<stdio.h>
int main( void )
{
    int x, y, z;
    int *int_addr;
    x = 41;
    y = 12;
    // int_addr x degiskenini isaret ediyor.
    int_addr = &x;
    // int_addr'in isaret ettigi degiskenin sakladigi deger aliniyor. (yani x'in degeri)
    z = *int_addr;
    printf( "z: %d\n", z );
    // int_addr, artik y degiskenini isaret ediyor.
    int_addr = &y;
    // int_addr'in isaret ettigi degiskenin sakladigi deger aliniyor. (yani y'nin degeri)
    z = *int_addr;
    printf( "z: %d\n" ,z );

    return 0;
}
```

Size of Pointers

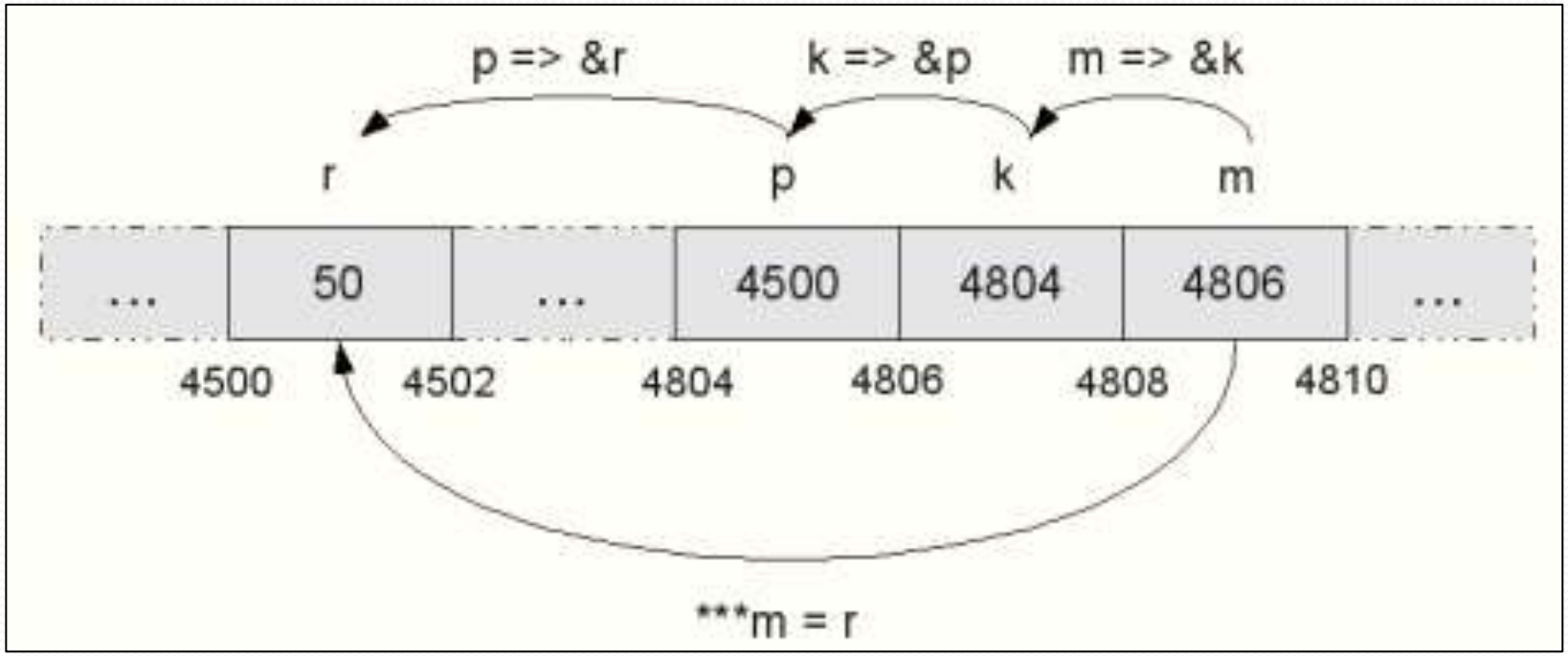
- Pointers generally have a fixed size, for ex. on a 32-bit executable they're usually 32-bit.

```
1  #include<stdio.h>
2  int main()
3  {
4      double i;
5      double *iptr;
6
7      iptr = &i;
8      printf("i boyutu: %d\n", sizeof(i));
9      printf("iptr boyutu: %d", sizeof(iptr));
10
11     getchar();
12     return 0;
13 }
```

Pointers that Points other Pointers

- As seen pointers stores the memory addresses of variables.
- Pointer is also a variable and an other pointer that shows a pointer can be defined.
- If we define a pointer variable that shows a pointer; we use '**' in the begining of pointer name.
- Number of * can change. If we define a pointer that points an other pointer that points an other pointer we have to use ‘***’.

Pointers that Points other Pointers



Pointer Arithmetic

- We can use increment, decrement (++,--), addition or subtraction operators with pointers. An integer have to be added or subtracted.
- When we increment the pointer by 1, pointer shows the next data block.
- New pointer value depends on the data type that pointer shows.

```
int i , *iPtr;
iPtr = &i; // Assume iPtr shows address 1000
iPtr += 2  // After this operation new value of  iPtr is 1008 (iPtr+2*4)
```

- Because int data type stored in 4 bytes in memory.

Pointer Arithmetic

```c
#include<stdio.h>
int main( void )
{
    int i, *iPtr;
    double y, *yPtr;

    iPtr = &i;
    printf("iPtr gosterdigi adres: %d \n", iPtr);
    iPtr ++;  //int tipi için bir sonraki adres bloğu 4 bayt fazlası.
    printf("iPtr gosterdigi adres: %d \n\n", iPtr);

    yPtr = &y;
    printf("yPtr gosterdigi adres: %d \n", yPtr);
    yPtr ++;  //double tipi için bir sonraki adres bloğu 8 bayt fazlası.
    printf("yPtr gosterdigi adres: %d ", yPtr);

    getchar();
    return 0;
}
```

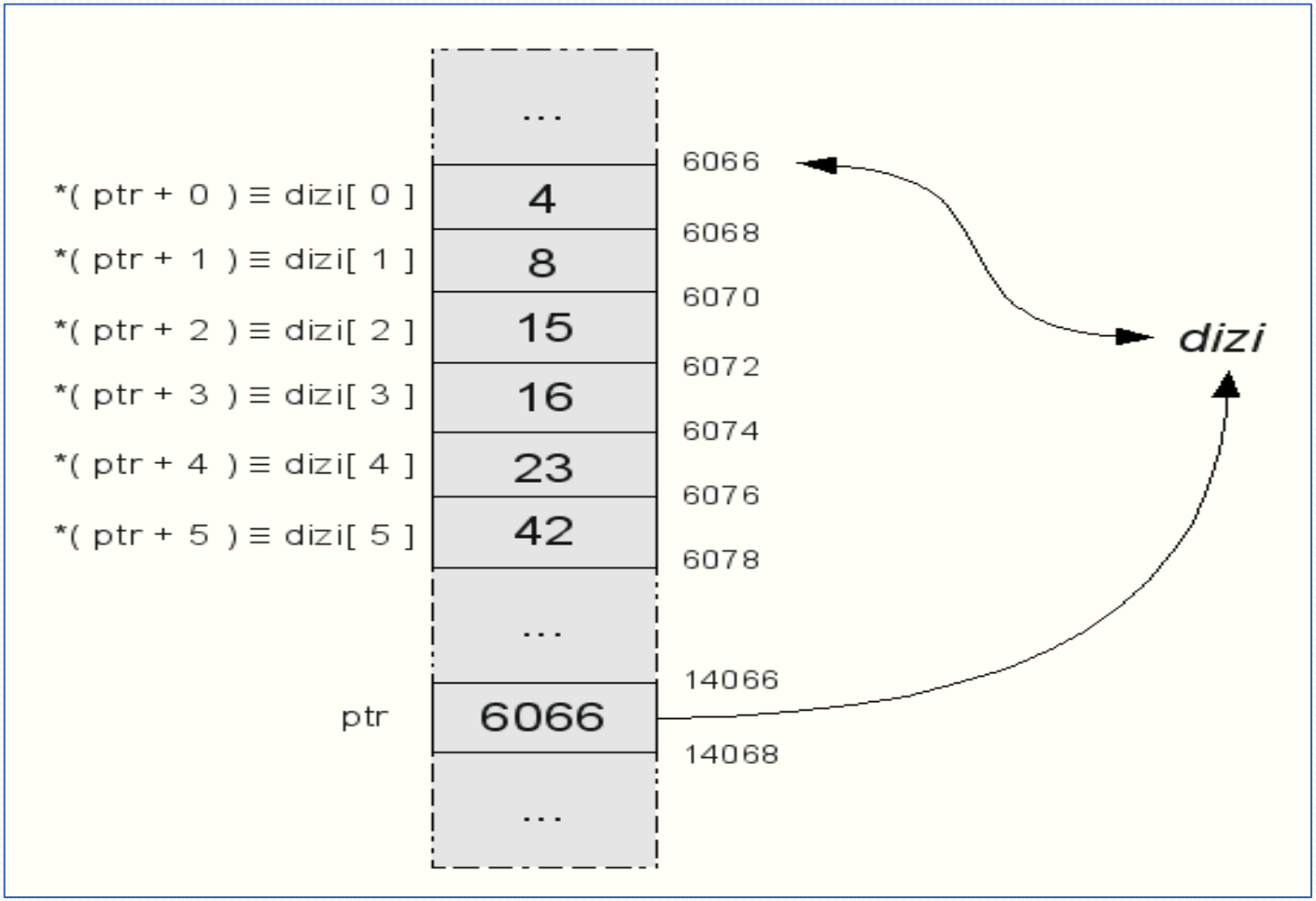
Pointer Arithmetic

- int i , *iPtr;
- iPtr = &i; // Assume iPtr shows address 1000
- **(*iPtr) ++; // Causes to increment value stored in the address 1000.**
- **iPtr ++; // Causes iPtr to show adress 1004 in memory**
- (*iPtr) +=2; // Causes to increment value by 2 stored in 1000 (*iPtr) =7; // Causes to assign 7 in address 1000.
- *(iPtr+2) = 5; //Causes to assign 5 in address 1008.

Relationship Between Pointers and Arrays

- An array name can be thought of as a constant pointer.
- Arrays and Pointers are closely related
- Pointers can also point arrays like they point variables.
  - int dizi [6];
  - int *ptr;
  - To set them equal use
    - The array name dizi actually is the adress of first element of the array dizi.
      - ptr = dizi; //Now ptr[0] and dizi[0] is same.
    - To explicitly assign ptr to the address of first element of dizi
      - ptr = & dizi[0];

Relationship Between Pointers and Arrays



Relationship Between Pointers and Arrays

- Pointers that shows arrays is generally used with
  - *(ptr + n) → where n indicates the index number of element in the array
  - *(ptr + 4) → gets the value of element dizi [4]
- Alternatives to access element dizi [4]
  - ptr[4]
  - *(dizi + 4)

Relationship Between Pointers and Arrays

```c
#include<stdio.h>
int main( void )
{
    int elm;
    int month[ 12 ];
    int *ptr;
    ptr = month; // month[0] 'ın adresini ptr'ye ata
    elm = ptr[ 3 ]; // elm = month[ 3 ]
    ptr = month + 3; // ptr, month[ 3 ] adresini göstersin
    ptr = &month[ 3 ]; // ptr, month[ 3 ] adresini göstersin
    elm = ( ptr+2 )[ 2 ]; // elm = ptr[ 4 ] ( = month[ 7 ] ).
    elm = *( month + 3 );
    ptr = month; // month[0] 'ın adresini ptr'ye ata
    elm = *( ptr + 2 ); // elm = month[ 2 ]
    elm = *( month + 1 ); // elm = month[ 1 ]

    return 0;
}
```

Relationship Between Pointers and Arrays

```c
#include <stdio.h>
int main()
{
    int i[10], j;
    int *iptr;
    for (j=0; j<10; j++)
        i[j]=j;

    iptr = i;
    for (j=0; j<10; j++) {
        printf("%d ", *iptr);
        iptr++;
    }
    /* iptr artık dizinin başını göstermez */
    printf("\n%d \n",*(iptr-1));
    iptr = i;
    for (j=0; j<10; j++)
        printf("%d ", *(iptr+j));
    /* iptr hala dizinin başını gösterir */
    printf("\n%d",*iptr);
    getchar();
    return 0;
}
```

Relationship Between Pointers and Arrays

```
 1  #include <stdio.h>
 2  int main()
 3  {
 4        char *a="1234567890";
 5        char x[10];
 6        char *p1, *p2;
 7        printf("%s\n", a);
 8        p1 = a;
 9        p2 = x;
10        while (*p1 != '\0') {
11             *p2 = *p1;
12              p1++;
13              p2++;
14        }
15        printf("%s\n", x);
16        getchar();
17        return 0;
18  }
```

Relationship Between Pointers and Arrays

- Arrays can contain pointers.
- Can access multiple arrays with arrays of pointers.
- We just assign the starting address of arrays to the arrays of pointers.
- Any modification you make on array of pointer will affect the original array.

Relationship Between Pointers and Arrays

```c
#include <stdio.h>
int main()
{
    int i,j;
    char * ilkBaharAylar[3] ={"Mart","Nisan","Mayis"};
    char * yazAylar[3] ={"Haziran","Temmuz","Agustos"};
    char * sonBaharAylar[3] ={"Eylul","Ekim","Kasim"};
    char * kisAylar[3] ={"Aralik","Ocak","Subat"};

    char ** table[4];//char pointer(string) tutan dizileri tutan dizi
    table[0] = ilkBaharAylar;
    table[1] = yazAylar;
    table[2] = sonBaharAylar;
    table[3] = kisAylar;

    for(i=0;i<4;i++)
    {
        for(j=0;j<3;j++)
        {
            printf("%s\n",table[i][j]);
        }
    }

    getchar();
    return 0;
}
```

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.
- 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 using pass by reference is inevitable.
- Because return keyword can only send one value out of function.
- For example, we want to write a division function that gives division 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 is returned by reference method.

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;
}
```