CENG 351 LAB 1
**ARRAY**

- is a data structure that stores a large number of elements in a fixed size sequential collection of the same type.
- You can declare individual variables instead of using arrays, but it is so hard to declare if you need to define so many variables. In this situation, defining array is more logical.
- Declaration of array is like this

```
  type   arrayName [ arraySize ];
```

-can be **int**, **char**, **double** etc.
ARRAY

- A specific element in a declared array is accessed by an index.
For example, you want to define an integer array which stores 10 integer numbers. You should define like this:

```c
int arrList[10];
```

`arrList[0]` shows the 1st element of the array and

`arrList[9]` shows the last element of the array.

The index of array starts always from 0.
Until this slide, we studied about one-dimensional arrays. Now, we start to study about two-dimensional arrays.

Declaration of two-dimensional array is like this:

```plaintext
type arrayName [rowSize] [columnSize];
```

- This number shows row number
- This number shows column number
**ARRAY**

- When we define a two-dimensional array like below statement,

```c
int arrList[5][10];
```

- this will create below table

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>12</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

arrList[3][7] = 12 This means 4th row and 8th column
There is no variable type as *string* in C language. You should define char array if you want to keep string expressions.

```c
char name[20];
```

**CHAR ARRAY = STRING**
In C programming, you cannot assign a string value like this:

```c
name = "özgür";
```

If you want to assign a string value, you should add "string.h" library in your project. After that, you can use the string functions.

```c
strcpy(name, "özgür");
```
In C programming, a struct is a declaration which consists of a list of variables under one name.

It can keep different variables from the different variable types, but the properties of the these variables should be closed to each other.

For example, we want to keep a book’s properties in a struct. Some properties of book:
STRUCT

- The definition of the book struct should be like this:

```c
#include <stdio.h>

struct book
{
    int pageNumber;
    char name[20];
    char writer[30];
};

int main()
{
    struct book b1;
    b1.
    
    name
    pageNumber
    writer
}
```

Definition of the struct

In order to use the struct, you have to create an object from this struct in the main function. After creating the object, you can reach the struct’s properties as you can see the left image.
There is an optional property in the struct structure. This property is "typedef". If you use this property, you don’t need to write "struct structName variableName" when you create a struct object in the main function. You can use like this instead of previous slide:

```c
#include <stdio.h>

typedef struct book
{
    int pageNumber;
    char name[20];
    char writer[30];
}Book;

int main()
{
    Book b1;
    b1.
}
```

If you use typedef, you should write struct name to here.

In the main function, it is enough to write just this name when you create an object from the struct.
You can also combine struct and array like below example:

```c
typedef struct book
{
    int pageNumber;
    char name[20];
    char writer[30];
} Book;

int main()
{
    Book bookList[5];
    for(int i = 0; i < 5; i++)
    {
        printf("Please enter the page number:");
        scanf("%d", &bookList[i].pageNumber);
        printf("Please enter the book's name:");
        scanf("%s", bookList[i].name);
        printf("Please enter the writer's name:");
        scanf("%s", bookList[i].writer);
    }
}
```

You can take a string value from the user like this. You should write “%s” to take string value. “%c” takes just a character, but “%s” takes whole characters until empty character.
A pointer is a data type that keeps another value’s address in itself. They point to locations in memory.

Let’s analyze the below example:

```c
#include <stdio.h>

int main()
{
    int a = 5;
    int *p;
    p = a;
    printf("%d", p);
    return 0;
}
```

In this line, you cannot assign `a` and `p` to each other, because the type of “a” variable is integer. And the type of “p” variable is POINTER integer. So, they are different than each other. The pointer’s value should be the address of the other variables. So you should write like this:

```c
p = &a;
```
If we write the below code, the table is created in memory.

```c
#include <stdio.h>

int main()
{
    int a = 5;
    int *p;
    p=&a;
    printf("%d\n", p);
    return 0;
}
```

<table>
<thead>
<tr>
<th>Address</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>00FF28</td>
<td>5</td>
</tr>
<tr>
<td>00FF29</td>
<td></td>
</tr>
<tr>
<td>00FF30</td>
<td></td>
</tr>
<tr>
<td>00FF31</td>
<td></td>
</tr>
<tr>
<td>00FF32</td>
<td></td>
</tr>
<tr>
<td>00FF33</td>
<td>00FF28</td>
</tr>
<tr>
<td>00FF34</td>
<td></td>
</tr>
<tr>
<td>00FF35</td>
<td></td>
</tr>
<tr>
<td>00FF36</td>
<td></td>
</tr>
</tbody>
</table>

“00FF28” is converted to integer number as “1899584”. If we write `%p` instead of `%d`, the result will be “00FF28”. 

Press any key to continue . . .
In order to write the value of “a” variable, we have to write like this:

```c
printf("%d\n", *p);
```

The important thing is that

ONE DIMENSIONAL ARRAY = POINTER

So we can use this statement

```c
int a[5] = {2,4,6,8,10};
int *p;
p=a;
```