OUTLINE

♦ Structures in C
♦ Declarations and Initializations of Structure Variables
♦ Accessing Structure Members: Dot and Arrow Operators

STRUCTURES

♦ Structures are collection of the related variables under one name. Structures may contain variables of many different data types.
♦ Structures are commonly used to define records, which contain information regarding a specific data object (mostly used in databases). Pointers and structures help the formation of more complex data structures such as linked lists, queues, stacks, trees and databases.

Ex: Consider a person who has the following personal student record:

<table>
<thead>
<tr>
<th>Name</th>
<th>John</th>
</tr>
</thead>
<tbody>
<tr>
<td>Surname</td>
<td>Smith</td>
</tr>
<tr>
<td>Age</td>
<td>19</td>
</tr>
<tr>
<td>Gender</td>
<td>M</td>
</tr>
<tr>
<td>CGPA</td>
<td>3.65</td>
</tr>
</tbody>
</table>

Then we can his record by:

```c
struct st_card {
    char name[15];
    char sname[15];
    int age;
    char gender;
    float cgpa;
};
```

Here keyword `struct` introduces the structure definition. The identifier `st_card` is the structure tag. Structure tag is used with keyword `struct` to declare variables of the structure type.

DECLARATION & INITIALISATION OF THE STRUCTURE VARIABLES.

Ex: `struct st_card a, class[30], *stptr;`

Alternatively:

```c
struct st_card {
    char name[15];
    char sname[15];
    int age;
    char gender;
    float cgpa;
};
```

♦ Initialisation of the structure variables.

```c
struct st_card a="John", "Smith", 19, 'M', 3.65;`

TYPEDEF (TYPE DEFINITION)

The keyword `typedef` provides a way of creating shorter data type names.
typedef struct {
    char name[15];
    char sname[15];
    int age;
    char gender;
    float cgpa;
} StCard;

StCard b, A[30], *ptr=&b;

StCard a={"Ayse","Uzun",26,'F',3.42}, class[30], *stptr=&a;

**ACCESSING MEMBERS OF STRUCTURES**

There are two operators used to access members of the structures.

- **Structure member operator (.)** dot operator.
- **Structure pointer operator (->)** arrow operator.

Ex:

```
typedef struct {
    char name[15];
    char sname[15];
    int age;
    char gender;
    float cgpa;
} StCard;

StCard a={"Ali","Veli",22,'M',3.88}, *aptr=&a;

printf("%s",a.name);  /* using dot operator */
printf("%d",a.age);
printf("%s",aptr->name); /* using arrow operator */
printf("%d",aptr->age);
```

**Ex:** A Traveling Agency keeps the records of the customers using the following set of data for each customer going on a holiday.

- Name of the customer
- Surname of the customer
- The number of days a customer spends on holiday
- The price for each day

(a) Create a new data type, CUSTOMERS, to represent the data structure of such a record.

```
type define struct {
    char name[15];
    char sname[15];
    int ndays;
}"
float price;
}CUSTOMERS;

(b) Write a function called ReadCustomers() to read all of the customer records in a season. Assume that the number of customers (size) is passed to the function as an argument.

void ReadCustomers(CUSTOMERS A[], int size)
{
    int i;
    for(i=0;i<size;i++){
        printf("Enter Customer Record:");
        scanf("%s", A[i].name);
        scanf("%s", A[i].sname);
        scanf("%d", &A[i].ndays);
        scanf("%f", &A[i].price);
    }
}

(c) Write another function called CustomerSpending() to calculate and display the name and surname as well as the total spending of each customer. The function will return the total spending of all the customers. Assume that the number of customers (size) is passed to the function as an argument.

float CustomerSpending(CUSTOMERS A[], int size)
{
    int i;
    float eachtotal, alltotal=0.0;
    for(i=0;i<size;i++){
        eachtotal = A[i].ndays* A[i].price;
        printf("Customer Name:%s, Surname:%s, Total Spending:%.2f\n", A[i].name, A[i].sname, eachtotal);
        alltotal=alltotal+ eachtotal;
    }
    return alltotal;
}