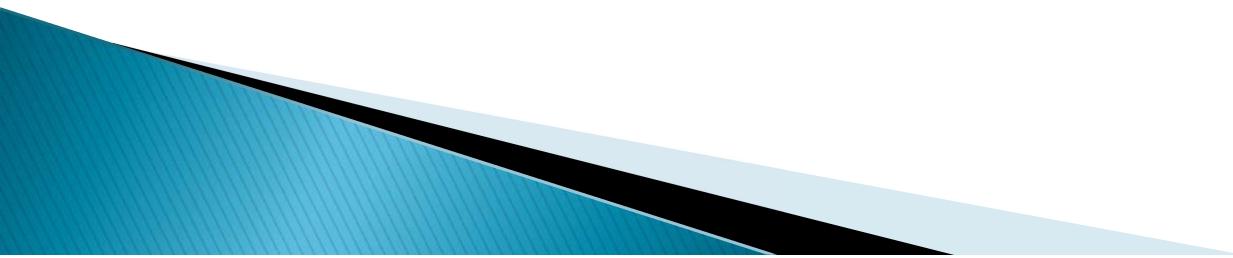


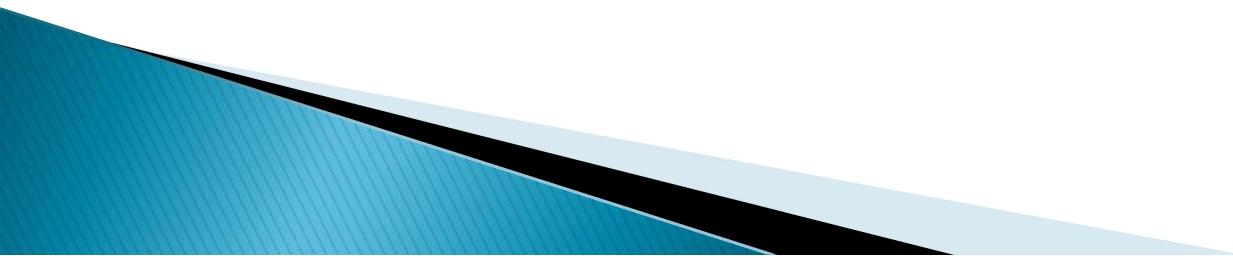
Structures

- ▶ The last major language facility in *C* to be introduced in this course.
- ▶ Essential for building up “interesting” data structures
 - e.g.,
 - Data structures of multiple values of different kinds
 - Data structures of indeterminate size



Definition — *Structure*

- ▶ A collection of one or more variables, typically of different types, grouped together under a single name for convenient handling
- ▶ Known as **struct** in C



What is a Structure?

- ▶ Structure is the collection of variables of different types under a **single name** for better handling.
- ▶ **For example:** You want to store the information about person about his/her name, citizenship number and salary.
- ▶ Keyword struct is used for creating a structure.

Using Structures

- ▶ Define the structure.
- ▶ Declare/Initialize instances of the structure.
- ▶ Access members of an instance of the structure.

Syntax of Structure

```
struct structure_name  
{ data_type member1;  
    data_type member2;  
    .  
    .  
    data_type memeber; };
```

struct

- ▶ Defines a new *type*
 - i.e., a new kind of data type that compiler regards as a unit

- ▶ E.g.,

```
struct motor {  
    float volts;           //voltage of the motor  
    float amps;            //amperage of the motor  
    int phases;             //# of phases of the motor  
    float rpm;              //rotational speed of motor  
};                      //struct motor
```

Structure Variable Declaration

- ▶ When a structure is defined, it creates a user-defined type but, no storage is allocated.
- ▶ For the structure of person, variable can be declared as:

Structure Variable Declaration

```
struct person  
{ char name[50];  
    int cit_no;  
    float salary;  
};
```

Inside main function:

```
struct person p1, p2, p[20];
```

```
struct person  
{ char name[50];  
    int cit_no;  
    float salary;  
}p1 ,p2 ,p[20];
```

Accessing Members of a Structure

- ▶ There are two types of operators used for accessing members of a structure:
 1. Member operator(.)
 2. Structure pointer operator(>) (will be discussed in structure and pointers)

Accessing Members of a Structure -1

- ▶ Any member of a structure can be accessed as:
`structure_variable_name.member_name`
- ▶ Suppose, we want to access salary for variable p2. Then, it can be accessed as:

p2.salary

struct

- ▶ Defines a new *type*
- ▶ E.g.,

```
struct motor {  
    float volts;  
    float amps;  
    int phases;  
    float rpm;  
};           //struct motor
```

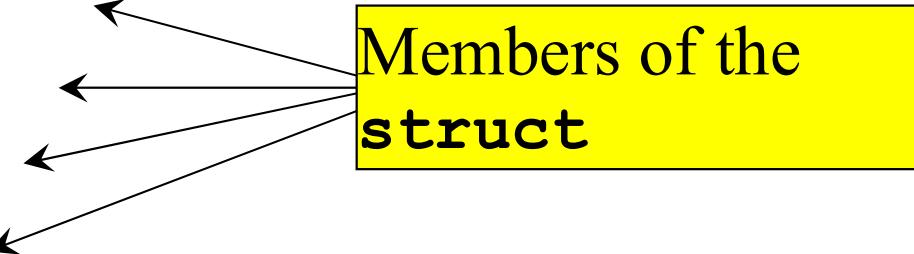
Name of the type

struct

- ▶ Defines a new *type*
- ▶ E.g.,

```
struct motor {  
    float volts;  
    float amps;  
    int phases;  
    float rpm;  
};           //struct motor
```

Members of the
struct



```
graph LR; A[Members of the struct] --> B[ ]; A --> C[ ]; A --> D[ ]; A --> E[ ];
```

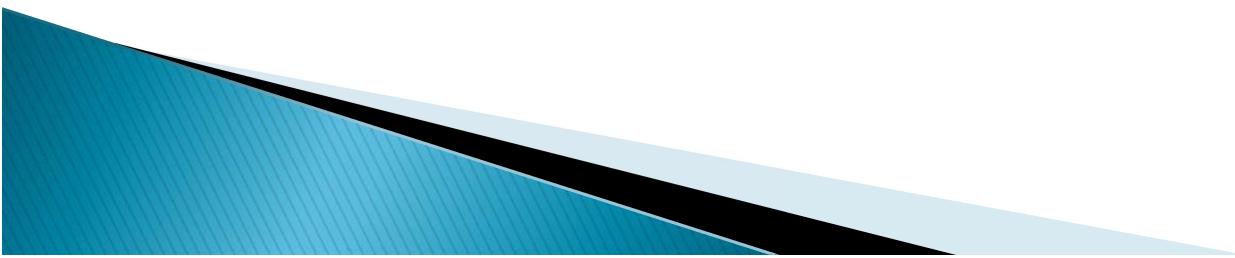
Declaring struct variables

```
struct motor p, q, r;
```

- Declares and sets aside storage for three variables – **p**, **q**, and **r** – each of type **struct motor**

```
struct motor M[25];
```

- Declares a 25-element array of **struct motor**; allocates 25 units of storage, each one big enough to hold the data of one **motor**



Accessing Members of a struct

▶ Let

```
struct motor p;  
struct motor q[10];
```

▶ Then

p.volts	— is the voltage
p.amps	— is the amperage
p.phases	— is the number of phases
p.rpm	— is the rotational speed
q[i].volts	— is the voltage of the i th motor
q[i].rpm	— is the speed of the i th motor

Operations on struct (continued)

- ▶ Remember:—
 - Passing an argument by value is an instance of *copying* or *assignment*
 - Passing a return value from a function to the caller is an instance of *copying* or *assignment*
- ▶ E.g,:—

```
struct motor f(struct motor g) {  
    struct motor h = g;  
    ...;  
    return h;  
}
```



Initialization of a struct



Typedef

- ▶ Definition:— a **typedef** is a way of *renaming* a type
- ▶ E.g.,

```
typedef struct motor Motor;
```

```
Motor m, n;
```

```
Motor r[25];
```

```
Motor function(const Mot
```

E.g., **typedef**, lets you
leave out the word “**struct**”

typedef (continued)

- ▶ **typedef** may be used to rename *any* type
 - Convenience in naming
 - Clarifies purpose of the type
 - Cleaner, more readable code
 - Portability across platforms
- ▶ E.g.,
 - `typedef int size_t;`
 - `typedef long int32;`
 - `typedef long long int64;`



Keyword **typedef**

- ▶ Programmer generally use **typedef** while using structure in C language. For example:

```
typedef struct complex  
{ int imag;  
float real; } comp;
```

Inside main:
comp c1,c2;

- ▶ Here, **typedef** keyword is used in creating a type **comp** (which is of type as **struct complex**). Then, two structure variables **c1** and **c2** are created by this **comp** type.

Structure and Function

- ▶ In C, structure can be passed to functions by two methods:
 1. Passing by value (passing actual value as argument)
 2. Passing by reference (passing address of an argument)

Passing Structure by Value

```
#include <stdio.h>
struct student{
char name[50];
int roll;};
void Display(struct student stu);
int main()
{struct student s1;
printf("Enter student's name: ");
scanf("%s",&s1.name);
printf("Enter roll number:");
scanf("%d",&s1.roll);
Display(s1);
return 0;}
void Display(struct student stu)
{ printf("Output\nName: %s",stu.name);
printf("\nRoll: %d",stu.roll);}
```

Output

Enter student's name: Kevin
Enter
roll number: 149

Output

Name: Kevin

Roll: 149

Enumeration -1

- ▶ Is a set of named integer constants that specifies all the legal values that a variable of its type can have.

```
enum color {red, white, blue}
```

```
color C;
```

```
C = red;
```

```
C = white;
```

Enumeration -2

- ▶ The key point to understand about an enumeration that each of the symbols stands for an integer value and can be used in any integer expression.

Enumeration -3

```
#include <stdio.h>
int main()
{
enum
    Days{Sunday,Monday,Tuesday,Wednesday,Thursday,Friday,Saturday};

Days TheDay;
int j = 0;
printf("Please enter day of the week (0 to 6)\n");  scanf("%d",&j);
TheDay = Days(j);

if(TheDay == Sunday || TheDay == Saturday)
    printf("Hurray it is the weekend\n");
else
    printf("still at work ");
return 0;
}
```

Unions

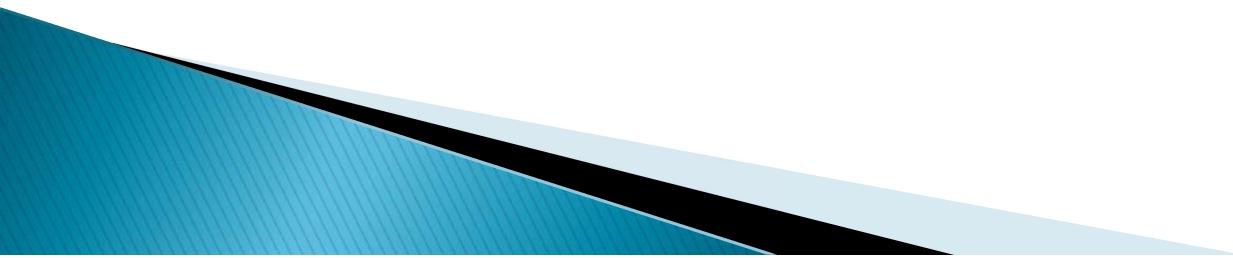
- ▶ A **union** is like a **struct**, but only one of its members is stored, not all
 - i.e., a single variable may hold different types at different times
 - Storage is enough to hold largest member
 - Members are overlaid on top of each other
- ▶ E.g.,

```
union {  
    int ival;  
    float fval;  
    char *sval;  
} u;
```



Unions (continued)

- ▶ **unions** are used much less frequently than **structs** — mostly
 - in the inner details of operating system
 - in device drivers
 - in embedded systems where you have to access registers defined by the hardware



Example

- ▶ Define a struct student that holds:
 - Name
 - GPA
 - Term
 - Enter the information for 5 students
 - Write a function that calculates the average GPA

Questions?

