C Programming

UNIT 3 Handouts / Class Notes

UNIT - 3 Part 2 Structures & Unions

Dr. Suresh Mudunuri, SRKR Engineering College

Unit 3 Syllabus

UNIT-III (10 Hrs)	Arrays: Concepts, Using Array in C, Array Application, Two Dimensional Arrays, Multidimensional Arrays, Programming Example – Calculate Averages Strings: String Concepts, C String, String Input / Output Functions, Arrays of Strings, String Manipulation Functions String/ Data Conversion, A Programming Example – Morse Code Enumerated, Structure, and Union: The Type Definition (Type def), Enumerated Types, Structure, Unions, and Programming Application.

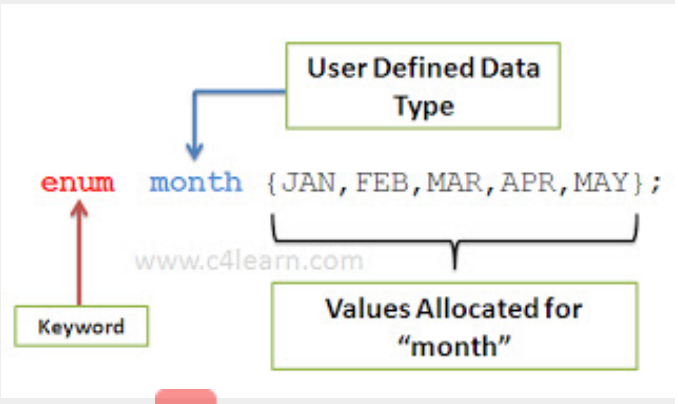
Enumerated Data Types

- Enumeration **identifiers can share the same value**. For example, in
  `enum switch { no=0, off=0, yes=1, on=1 };`
  the identifiers no and off to an `enum switch variable` both have the **value 0**.
  And the identifiers `yes and on both` have the **value 1**.

- The variations permitted when defining an enumerated data type are similar to those permitted with structure definitions:
  - ✓ The name of the data type can be omitted, and
  - ✓ variables can be declared to be of the particular enumerated data type when the type is defined.

- As an example showing both of these options, the statement
  `enum { east, west, south, north } direction;`
  defines an (unnamed) enumerated data type with values `east, west, south, or north`, and **declares a variable direction to be of that type**.

```c
#include <stdio.h>

enum week { sunday, monday, tuesday, wednesday, thursday, friday, saturday };

int main()
{
    enum week today;
    today = wednesday;
    printf("Day %d",today+1);
    return 0;
}
```



Typedef

Typedef can be used to create synonyms for previously defined data type names.

For example,

typedef int Length;

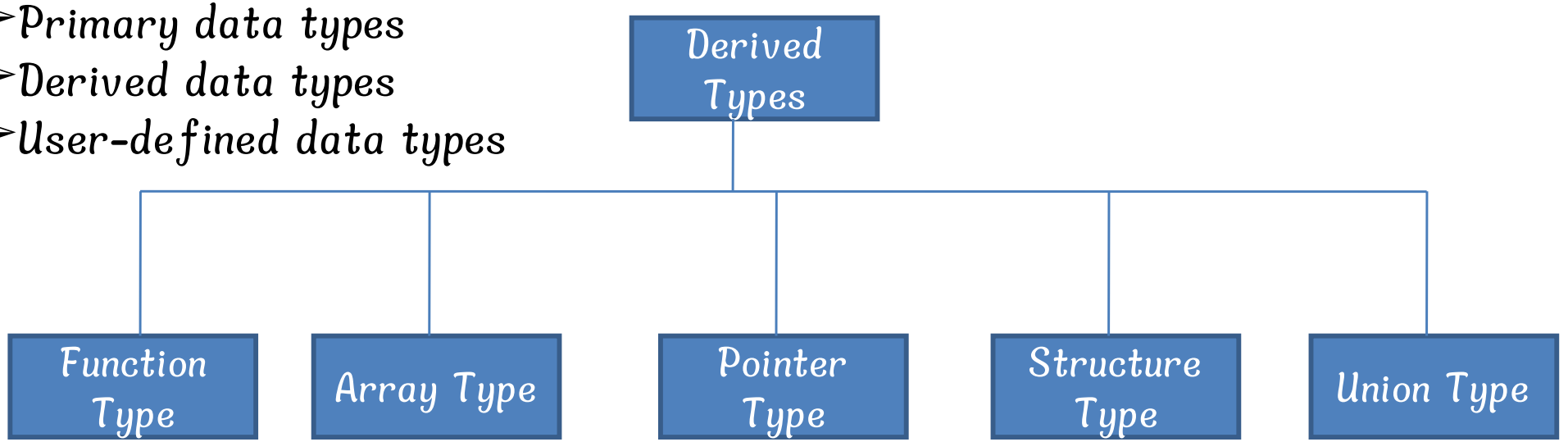
Makes length a synonym for integer.

DEMO CODE

```c
#include <stdio.h>

typedef int age;
typedef float real;

void main()
{

 age a,b;
 real f=14.3;

 a=20;
}
```

C Data Types:

- Primary data types
- Derived data types
- User-defined data types



Array – Collection of one or more related variables of similar data type grouped under a single name

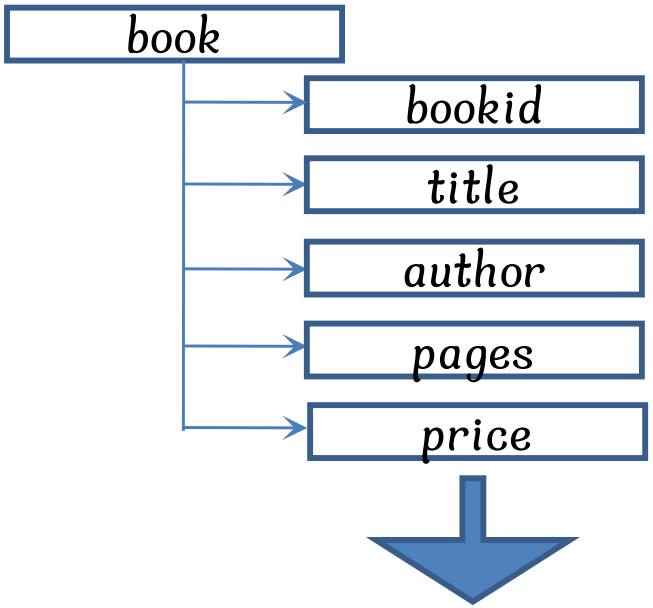
Structure – Collection of one or more related variables of different data types, grouped under a single name

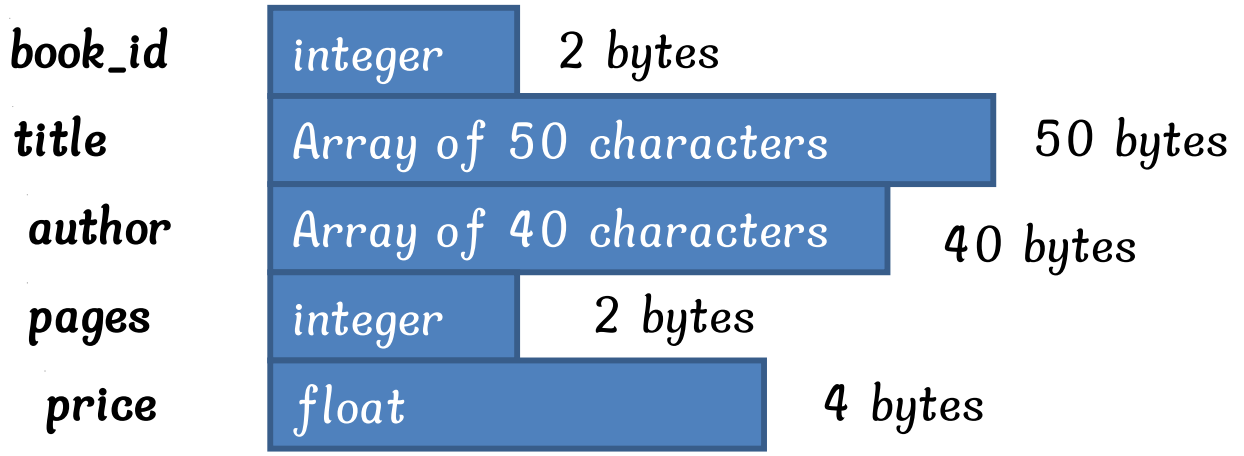
In a Library, each book is an **object**, and its **characteristics** like title, author, no of pages, price are grouped and represented by one **record**.

The characteristics are different types and grouped under a aggregate variable of different types.

A **record** is group of **fields** and each field represents one characteristic. In C, a record is implemented with a derived data type called **structure**. The characteristics of record are called the **members** of the structure.

Book-1	Book-2	Book-3
BookID: 1211	BookID: 1212	BookID: 1213
Title : C Primer Plus	Title : The ANSI C Programming	Title : C By Example
Author : Stephen Prata	Author : Dennis Ritchie	Author : Greg Perry
Pages : 984	Pages : 214	Pages : 498
Price : Rs. 585.00	Price : Rs. 125.00	Price : Rs. 305.00





Memory occupied by a Structure variable

STRUCTURE- BOOK

```
struct   book  {
         int  book_id ;
         char title[50] ;
         char author[40] ;
         int pages ;
         float price ;
};
```

Structure tag

```
struct   < structure_tag_name >
 {
         data type < member 1 >
         data type < member 2 >
              .... .... .... ....
         data type < member N >
} ;
```

Declaring a Structure Type

```
struct student
{
        int roll_no;
        char name[30];
        float percentage;
};
```

Declaring a Structure Variable

```
struct student s1,s2,s3;
        (or)
struct student
{
        int roll_no;
        char name[30];
        float percentage;
}s1,s2,s3;
```

Initialization of structure

Initialization of structure variable while declaration :

```
struct student s2 = { 1001, " K.Avinash ",
                      87.25 } ;
```

Initialization of structure members individually :

```
s1. roll_no = 1111;
strcpy ( s1. name , " B. Kishore " ) ;
s1.percentage = 78.5 ;
```

membership operator

Reading values to members at runtime:

```
struct student s3;
printf("\nEnter the roll no");
scanf("%d",&s3.roll_no);
printf("\nEnter the name");
scanf("%s",s3.name);
printf("\nEnter the percentage");
scanf("%f",&s3.percentage);
```

Implementing a Structure

```
struct employee {
        int empid;
        char name[35];
        int age;
        float salary;
};
int main()  {
      struct employee emp1,emp2 ;

      struct employee emp3 = { 1213 , " S.Murali " , 31 , 32000.00 } ;
      emp1.empid=1211;
      strcpy(emp1.name, "K.Ravi");
      emp1.age = 27;
      emp1.salary=30000.00;
      printf("Enter the details of employee 2");
      scanf("%d %s %d %f " , &emp2.empid, emp2.name, &emp2.age, &emp2.salary);
      if(emp1.age > emp2.age)
            printf( " Employee1 is senior than Employee2\n" );
      else
            printf("Employee1 is junior than Employee2\n");

      printf("Emp ID:%d\n  Name:%s\n  Age:%d\n    Salary:%f",    emp1.empid,emp1.name,emp1.age,
emp1.salary);
}
```

Declaration of Structure Type

Declaration of Structure variables

Declaration and initialization of Structure variable

Initialization of Structure members individually

Reading values to members of Structure

Accessing members of Structure

Arrays And structures

```
struct student
{
    int sub[3] ;
    int total ;
} ;


int  main( )  {
   struct student s[3];
   int i,j;
    for(i=0;i<3;i++) {
        printf("\n\nEnter student %d marks:",i+1);
        for(j=0;j<3;j++) {
            scanf("%d",&s[i].sub[j]);
        }
    }
    for(i=0;i<3;i++) {
          s[i].total =0;
          for(j=0;j<3;j++) {
                s[i].total +=s[i].sub[j];
          }
          printf("\nTotal marks of student %d is: %d",
                                i+1,s[i].total );
    }
}
```

Nesting of structures

```
struct date {
    int day ;
    int month ;
    int year ;
} ;
struct  person {
    char  name[40];
    int  age ;
    struct  date  b_day ;
};
int  main( )  {
   struct  person  p1;
   strcpy ( p1.name , "S. Ramesh " ) ;
   p1. age =  32 ;
   p1.b_day.day  = 25 ;
   p1.b_day. month = 8 ;
   p1.b_day. year = 1978 ;
}
```

Outer Structure

Inner Structure

Accessing Inner Structure members

OUTPUT:

```
Enter student 1 marks: 60 60 60
Enter student 2 marks: 70 70 70
Enter student 3 marks: 90 90 90

Total marks of  student 1 is: 180
Total marks of  student 2 is: 240
Total marks of  student 3 is: 270
```

structures and functions

```
struct   fraction  {
     int numerator ;
     int denominator ;
};

void  show ( struct fraction f);
int  main ( ) {
     struct fraction f1  =  {  7, 12 } ;
     show  ( f1 ) ;
}

void  show ( struct fraction f  )
{
   printf ( " %d / %d ", f.numerator,
                    f.denominator ) ;
}
```

OUTPUT: 7 / 12

Self referential structures

```
struct student_node  {
    int roll_no ;
    char  name [25] ;
    struct student_node  *next ;
} ;
int  main( )
 {
   struct  student_node  s1 ;
   struct  student_node  s2 = { 1111, "B.Mahesh", NULL } ;
   s1. roll_no  = 1234 ;
   strcpy ( s1.name , "P.Kiran " ) ;

    s1. next  =  & s2 ;

    printf ( " %s ", s1. name  ) ;

    printf ( " %s " , s1.next - > name  ) ;
}
```

s2 node is linked to s1 node

Prints P.Kiran

Prints B.Mahesh

A self referential structure is one that includes at least one member which is a pointer to the same structure type.

With self referential structures, we can create very useful data structures such as linked -lists, trees and graphs.

Pointer to a structure

```
struct product
{
    int prodid;
    char name[20];
};
int main()
{
    struct product inventory[3];
    struct product  *ptr;
    printf("Read Product Details : \n");
    for(ptr = inventory;ptr<inventory +3;ptr++) {
       scanf("%d %s", &ptr->prodid, ptr->name);
    }
    printf("\noutput\n");
    for(ptr=inventory;ptr<inventory+3;ptr++)
    {
       printf("\n\nProduct ID :%5d",ptr->prodid);
       printf("\nName :  %s",ptr->name);
    }
}
```

Accessing structure members through pointer :

i) Using . (dot) operator :

```
( *ptr ) . prodid = 111 ;
strcpy ( ( *ptr ) . Name, "Pen") ;
```

ii) Using - > (arrow) operator :

```
ptr - > prodid = 111 ;
strcpy( ptr - > name , "Pencil") ;
```

```
Read Product Details :

111 Pen
112 Pencil
113 Book

Print Product Details :

Product ID : 111
Name : Pen
Product ID : 112
Name : Pencil
Product ID : 113
Name : Book
```

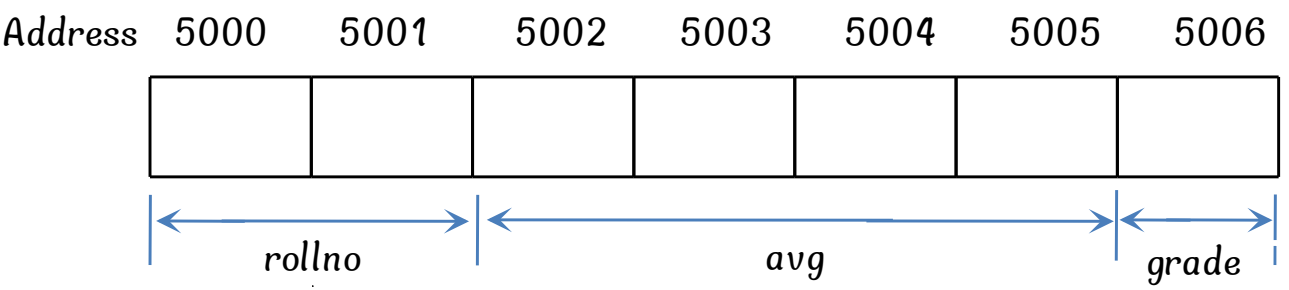
A union is a structure all of whose members share the same memory

Union is a variable, which is similar to the **structure** and contains number of members like structure.

In the structure each member has its own memory location whereas, members of union share the same memory. The amount of storage allocated to a union is sufficient to hold its largest member.

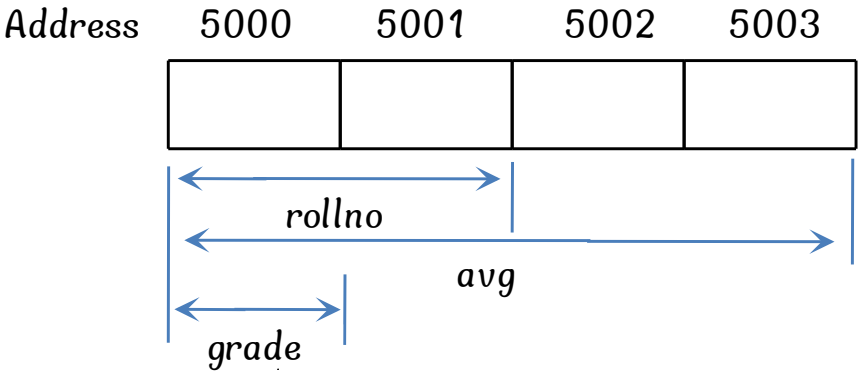
```
struct student {
    int rollno;
    float avg;
    char grade;
};
union pupil {
    int rollno;
    float avg;
    char grade;
};
int main() {
    struct student s1;
    union pupil p1;
    printf ("%d bytes",
        sizeof (struct student));
    printf ("%d bytes",
        sizeof (union pupil));
}
Output :
  7 bytes  4 bytes
```

Memory allotted to structure student



Total memory occupied : 7 bytes

Memory allotted to union pupil



Total memory occupied : 4 bytes