**Structure**

* Structures—sometimes referred to as aggregates—are collections of related variables under one name.
* **struct card {  
   char \*face;  
   char \*suit;  
  };**
* A structure cannot contain an instance of itself. A pointer to struct , however, may be included. A structure containing a member that is a pointer to the same structure type is referred to as a self-referential structure.
* **struct card aCard, deck[ 52 ], \*cardPtr; //** declares aCard to be a variable of type struct card, declares deck to be an array with 52 elements of type struct card and declares cardPtr to be a pointer to struct card.
* Alternatively:-

**struct card {  
 char \*face;  
 char \*suit;  
} aCard, deck[ 52 ], \*cardPtr;**

* Initialization:- **struct card aCard = { "Three", "Hearts" };**
* Accessing:- Two operators are used to access members of structures: the structure member operator (.)—also called the dot operator—and the structure pointer operator (->)—also called the arrow operator. Eg:- **printf( "%s", aCard.suit ); /\* displays Hearts \*/**
* The structure pointer operator—consisting of a minus (-) sign and a greater than (>) sign with no intervening spaces—accesses a structure member via a pointer to the structure. Eg- **printf( "%s", cardPtr->suit ); and cardPtr->suit is equivalent to (\*cardPtr).suit**

**Enumeration:-**

* An enumeration, introduced by the keyword enum, is a set of integer enumeration constants represented by identifiers.
* Values in an enum start with 0, unless specified otherwise, and are incremented by 1.
* For example, the enumeration

**enum months { JAN, FEB, MAR, APR, MAY, JUN, JUL, AUG, SEP, OCT, NOV, DEC };** creates a new type, enum months, in which the identifiers are set to the integers 0 to 11, respectively.

* To number the months 1 to 12, use the following enumeration:

**enum months {   
 JAN = 1, FEB, MAR, APR, MAY, JUN, JUL, AUG,   
 SEP, OCT, NOV, DEC };**

**Union:-**

* A union is a derived data type—like a structure—with members that share the same storage space
* For different situations in a program, some variables may not be relevant, but other variables are—so a union shares the space instead of wasting storage on variables that are not being used.
* The members of a union can be of any data type.
* The number of bytes used to store a union must be at least enough to hold the largest member.
* The union definition

**union number {**

**int x;**

**double y**

**};** indicates that number is a union type with members int x and double y

* Initialization:- **union number value = { 10 };**