STRUCTURE, UNION & BIT MANIPULATION

User-defined Data Types	Derived Data Types
· To create new data structures	· To extend or manipulate existing types.
· Defined by the programmer	· Derived from existing-types
· Memory usage can vary (eg. unions share memory).	· Memory usage is consistent with base data-types
eg. struct, union, enum, typedef	eg. array, pointer, function.

Structs: Data structures that can store combinations of characters, integer, floating pt. & enumerated type data.

A structure is a user-defined data type that groups related variables of different data types under a single name.

SYNTAX:

```
struct struct_name {
    datatype1 var1;
    datatype2 var2;
    ... };
struct struct_name s1;
```

struct struct_name defines a structure named struct_name. s1 is a variable of this type with var1, var2 etc. fields.

2 ways to declare variables of a struct :

①
```
struct Student {
    char name[50];
    int age;
    float grade;
} s1;      /* we declare a variable s1*/
```

②
```
struct Student{
    char name [50];
    int age;
    float grade; };
```

File to Refer: struct1.c

```
int main(){
  struct Student s1;
}
```

Accessing structure members:-

```
  printf("Age: %d", s1.age);
  s1.age = 20;
  printf ("Age : %d  ", s1.age);
```

OUTPUT:

```
Age : 0
Age : 20
```

Typedef

→ For creating synonyms of previously defined data type names.

eg. typedef int length;

'length' becomes a synonym for the datatype int.

eg. length a, b, len;
    length arr[10];

→ typedef is used in combination with struct to declare a synonym for a structure.

```
typedef struct Students{
    int rollno;
    char name[20]; } Student ;
```

// We can use student like any other type.

```
int main(){
Student stud 1;}
```

Files to Refer : struct2.c
struct3.

```
typedef struct Students{
    int roll;
    char name[5]; } student;

int main(){
  //Student stud1;              } Both
  struct Students stud1;        } work
  Stud1.roll=1;
  strcpy(stud1.name,"John");
  printf("%d\n", stud1.rollno);
  printf("%s\n", stud1.name);}
```

OR: typedef struct{ ← works w/o Students also!

```
          ...} student;
    int main(){
       Student stud1;
       :
       }
```

OUTPUT:
1
John

Array of Structures

```
Typedef struct Students{
    int rollno;
    char name[5];} Student;
```

File to Refer : struct 4.c

```
int main(){
    Student stud[2];       // OR  struct Students stud[2];
    :
    :  }
```

- The above declares an array of size 2.
- As any other array, the storage of memory is contiguous.

```
struct Students{
    int roll;
    char name[50];} stud[2];
(w/o using typedef)
```

Can refer : union3.c

Pointers to Structures

It is similar to declaring ptrs to Integers, double etc.

```
int *p1;
float *p2;
Student* p3;
```

ptr p3 stores address to a struct.

We can access values as: (*p3). rollno

'->' : shorthand operator (*p3). rollno ; } same meaning
p3->rollno;

File to refer: struct5.c

```
typedef struct Students{
    int roll;
  char name[10];} Student;

int main(){
   Student stud1;        // OR: struct Students stud1;
   stud1. roll = 1;
   strcpy(stud1.name, "John");
   Student* p1;        // OR: struct Students *p1;
   ~~printf("%d\n", (*p1).~~
Imp => p1 = &stud1;
   printf("%d\n", (*p1). roll);
     "     ("%s\n", (*p1). name);
     "    ("%d\n", p1 -> roll);
     "    ("%s\n", p1->name); }
```

```
OUTPUT: 1
        John
        1
        John
```

Unions

Similar to a structure but shares memory for all its members-

SYNTAX:

```
union unionName{

    dataType1  member1;
    dataType2  member2;
    ---
 }
```

Useful when variables don't need to exist simultaneously, conserving memory.

eg.
```
union Data{
    int intVal;
    float floatVal;
    char charVal; };
```

Files to Refer : union1.c
union2.c

```
union Data d1;
```

'union Data' defines a union with an integer, float and char sharing the same memory location.

```
union Student{
   int roll;
   char name[50];};

int main(){
  union Student student;
  student.roll=1;
  strcpy(student.name,"Mary");}
```

OR

```
union Student{
   int roll;
   char name[50];}d1;

int main(){
  d1.roll=1;
  strcpy(d1.name,"Mary);}
```

Real example on how to use union:

Files to Refer: union3.c

	STRUCTURE	UNION
Memory	Each member has unique space	All members share space
Size	Sum of sizes of all members	Size of largest number
Usage	To hold multiple data points	Save memory for single data usage at different times.