Chapter-10
Structure and Union

Structure:

- A structure is a user-defined data type available in C that allows to combining data items of different kinds.
- Structures are used to represent a record.
- All the structure elements are stored at contiguous memory locations.
- Structure type variable can store more than one data item of varying data types under one name.
- Structure is defined using "struct" keyword.

Syntax to define structure:

```
struct tag_name/structure_name
{
  member-1;
  member-2
};
```

Example:
Structure definition for storing student data

```
struct Student
{
   int roll_no;
   char name[20];
   int fees;
};
```

Here,

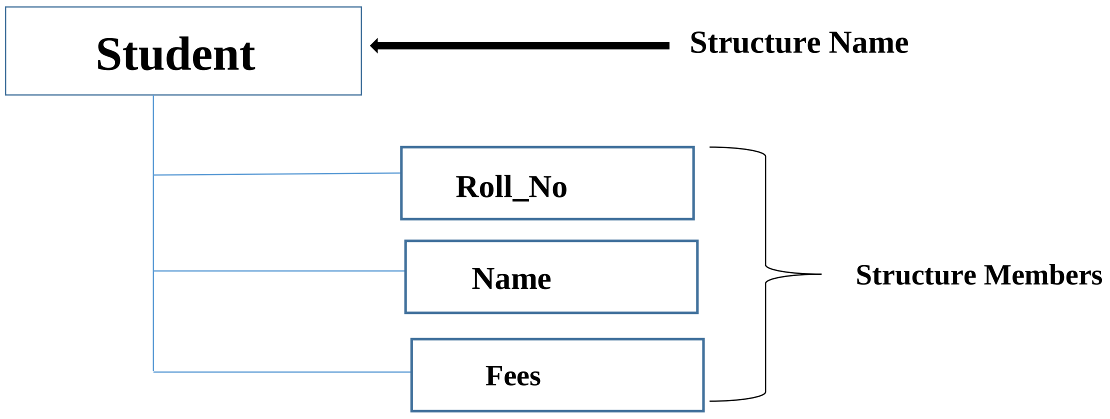
- **struct** is a keyword.
- **Student** is name of the structure and it is also called as structure tag name.
- **Roll_no,name and fees** are the fields of the structure which is known as structure members.
- There should be semicolon at the end of closing brace.

Note:
The structure definition does not reserve any space in memory for the members. So, it is called a structure template and memory will not be allocated for the template. Memory is allocated for the variables. This is pictorially represented as shown below for above defined structure student:

- **roll no -> int type = 4 bytes**
- **name[20]-> char type array =20 bytes**
- **fees -> int type = 4 bytes**

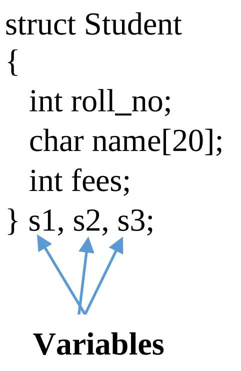
Total=28 bytes

so whenever we will declare variable of student type it will take 28 bytes as per above definition of student structure.



Structure variable declaration:
Structure variables are used to access the members of structure. There are 2 ways to create a structure variable.

1st way:



```
struct Student
{
  int roll_no;
  char name[20];
  int fees;
} s1, s2, s3;
```

Here, s1, s2, s3 are the structure variables.

2nd way:

```
struct Student
{
  int roll_no;
  char name[20];
  int fees;
};
int main()
{
   struct Student s1,s2;  //here s1 and s2 are the variables
   return 0;
}
```

Structure Initialization

- Initializing structure variables is similar to that of array i.e. all the elements will be enclosed within curly braces { } and are separated by comma (,).
- structure variables can be initialized in following ways:

1st way

```
struct student
{
     int roll_no;
     char name[20];
     int fees;
```

$$\} \underset{\substack{\uparrow\\ \textbf{Variable}}}{s1}=\{\underbrace{10,\text{“Neha”},54000}_{\textbf{values of variable s1}}\},\underset{\substack{\uparrow\\ \textbf{Variable}}}{s2}=\{\underbrace{12,\text{”priya”},60000}_{\textbf{Values of variable s2}}\};$$

2nd way

```
struct Student
{
   int roll_no;
   char name[20];
   int fees;
};
int main()
{
    struct Student s1={19,”seema”,89000};
    return 0;
}
```

3rd way

```
struct Student
{
   int roll_no;
   char name[20];
   int fees;
};
int main()
{
    struct Student s1;
    e1.roll_no=76;
    strcpy(e1.name,”Priyanka”);
    e1.fees=34000;
    return 0;
}
```

Accessing structure members

A member of a structure can be accessed using member access operator (.) dot, which in turn is followed by member name**.**

Syntax

variable_name **.** member_name;

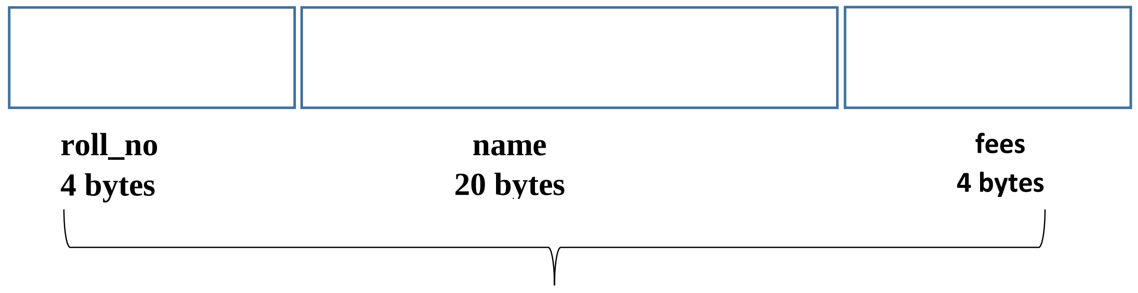
Example: Program to store student information using structure and display it.

```
#include<stdio.h>
struct student
{
   int roll_no;
   char name[20];
   int fees;
}s1={76,"Priyanka",65000};
int main()
{
    printf("Student information are\n");
    printf("Name:%s",s1.name);
    printf("\nRoll no:%d",s1.roll_no);
    printf("\nFees:%d",s1.fees);
    return 0;
}
```

Output:

```
Student information are
Name:Priyanka
Roll no:76
Fees:65000
```

Memory Representation



Memory for variable s1
s1 will occupy 4+20+4=28 bytes

Program to display employee record and input from user for structure variable.

```
#include<stdio.h>
struct emp
{
   int eid;
   char name[20];
   char desig[20];
   int salary
};
int main()
{
    struct emp e;
    printf("Enter employee id:");
    scanf("%d",&e.eid);
    printf("Enter employee name:");
    scanf("%s",&e.name);
    printf("Enter employee designation:");
    scanf("%s",&e.desig);
   printf("Enter employee salary:");
scanf("%d",&e.salary);
printf("\nEmployee Details are\n");
 printf("\nId:%d",e.eid);
 printf("\nName:%s",e.name);
  printf("\nDesignation:%s",e.desig);
  printf("\nsalary:%d",e.salary);
    return 0;
}
```

Output:

```
Enter employee id:1001
Enter employee name:John
Enter employee designation:Manager
Enter employee salary:45000


Employee Details are:
```

```
Id:1001
Name:John
Designation:Manager
salary:45000
```

typedef

typedef is a keyword used in C language to assign alternative names to existing data types. Its mostly used with user defined datatypes but can also be used for built in data types, when names of the data types become slightly complicated to use in programs. Following is the general syntax for using typedef,

Syntax:

typedef <existing_name> <alias_name>

Example:

typedef unsigned long ulong;

The above statement define a term ulong for an unsigned long datatype. Now this ulong identifier can be used to define unsigned long type variables.

ulong i, j;

Program for typedef

```
#include<stdio.h>
int main()
{
        typedef int I;
        I num;
        num=10;
        printf("%d",num);
        return 0;
}
```

Output:

```
10
```

Program

```
#include<stdio.h>

typedef struct employee

{

   char name[50];

   int salary;

}emp; //emp is alias name of employee type
```

```
int main()
{
    emp e1;
    printf("\nEnter Employee record:\n");
    printf("\nEmployee name:\t");
    scanf("%s", e1.name);
    printf("\nEnter Employee salary: \t");
    scanf("%d", &e1.salary);
    printf("\nName is: %s", e1.name);
    printf("\nSalary is: %d", e1.salary);
}
```

Output:

```
Enter Employee record:
Employee name:  John
Enter Employee salary:  34000

Name is: John
Salary is: 34000
```

Program

```
#include<stdio.h>
struct employee
{
    char name[50];
    int salary;
};

int main()
{
    typedef struct employee emp; //another way to use typedef with structure
    emp e1;
    printf("\nEnter Employee record:\n");
    printf("\nEmployee name:\t");
    scanf("%s", e1.name);
    printf("\nEnter Employee salary: \t");
    scanf("%d", &e1.salary);
    printf("\nName is: %s", e1.name);
    printf("\nSalary is: %d", e1.salary);
}
```

Array of Structure:

- As we create array of int, float, char etc. similarly we can create array of structure too.
- Suppose we want to store information of 5 students then instead of declaring 5 different variables, we can create array of structure.

Program:

```
#include<stdio.h>
#include <string.h>
struct student
{
int rollno;
char name[10];
};
int main()
{
int i;
struct student st[5];
printf("Enter Records of 5 students\n");
for(i=0;i<5;i++)
{
printf("Enter Rollno:");
scanf("%d",&st[i].rollno);
printf("Enter Name:");
scanf("%s",&st[i].name);
}
printf("\nStudent Information List:");
for(i=0;i<5;i++)
{
printf("\nRollno:%d, Name:%s",st[i].rollno,st[i].name);
}
   return 0;
}
```

Output:

```
Enter Records of 5 students
Enter Rollno:10
Enter Name:Jenny
Enter Rollno:11
Enter Name:Riya
Enter Rollno:12
Enter Name:Sam
Enter Rollno:13
Enter Name:Seema
Enter Rollno:14
Enter Name:Neha
```

```
Student Information List:
Rollno:10, Name:Jenny
Rollno:11, Name:Riya
Rollno:12, Name:Sam
Rollno:13, Name:Seema
Rollno:14, Name:Neha
```

Structures and functions

Various of passing structure to a function

- **By passing individual member of structure**
- **By passing whole member structure**
- **By passing structure through pointers**

1. **By passing individual member of structure**

   A function can be called by passing the member of a structure as an actual parameter.

Program

```
#include<stdio.h>
#include<string.h>
struct book
{
        int bid;
   char name[20];
   int price;
};
 void data(char[]);
int main()
{
         struct book b;
    b.bid=1001;
    strcpy(b.name,"C Programming");
    b.price=235;

    data(b.name); //passing name member as parameter
}
void data(char n[20])
{
    printf("Book name is %s",n);
}
```

Output:

```
Book name is C Programming
```

2. **By passing whole member structure**
   While passing the whole structure just we need to pass the variable name.
   Program

```
#include<stdio.h>
#include<string.h>
struct book
{
      int bid;
    char name[20];
    int price;
};
 void data(struct book);
int main()
{
       struct book b;
     b.bid=1001;
     strcpy(b.name,"C Programming");
     b.price=235;

     data(b);
}
void data(struct book b1)
{
      printf("\nBook id is %d",b1.bid);
      printf("\nBook name is %s",b1.name);
      printf("\nBook price is %d",b1.price);
}
```

   Output

```
Book id is 1001
Book name is C Programming
Book price is 235
```

Union

- A union is a user-defined type similar to structs in C except for one key difference.
- Structure allocate enough space to store all its members whereas unions allocate the space to store only the largest member.
- It is defined using union keyword.
- In union all members share the same memory.

Defining union:

```
union union_name
{
  member-1;
  member-2
  member-n;
};
```

Example:

```
union Car
{
   char name[20];
   int price;
};
```

Create union variables

- When a union is defined, it creates a user-defined type. However, no memory is allocated.
- To allocate memory for a given union type and work with it, we need to create variables.

1st way:

```
  union car
  {
    char name[50];
    int price;
  } car1, car2;
```

2nd way:

```
  union car
  {
    char name[50];
    int price;
  };

  int main()
  {
    union car car1, car2;
    return 0;
  }
```

Here,

- car1 and car2 are variables of car type.
- members of car type are name and price, name variable will occupy 50 bytes whereas price will occupy 4 bytes,so the variable car1 and car2 will occupy 50 bytes, because the size of a union variable will always be the size of its largest member. In the above example, the size of its largest member (name[50]), is 50 bytes.

Assigning value in union variable

In union only one member can be initialized at once.

Correct way:

```
union Car
{
   char name[20];
```

```
  int price;
 }c;
 int main()
 {
  strcpy(c.name,” Ferrari “);
 c.price=500000;
 }
```

Incorrect way:

```
union Car
{
  char name[20];
  int price;
 }c={“Ferrari”,400000};
```

Incorrect way:

```
union Car
{
  char name[20];
  int price;
 }c;
 int main()
 {
  union car c={“Ferrari”,500000);
  return 0;
 }
```

Program for union

```
 #include <stdio.h>
union Job
 {
  float salary;
  int workerNo;
} j;

int main()
{
  j.salary = 12.3;

  // when j.workerNo is assigned a value,
  // j.salary will no longer hold 12.3
  j.workerNo = 100;
```

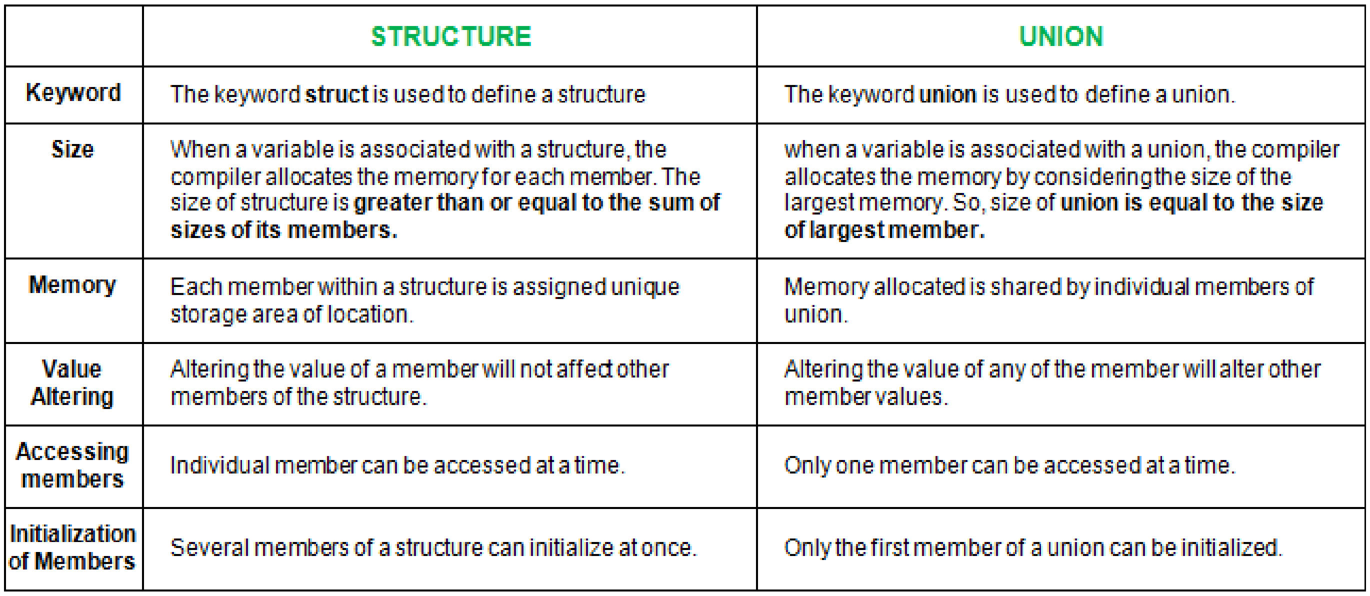
```
   printf("Salary = %.1f\n", j.salary);
   printf("Number of workers = %d", j.workerNo);
   return 0;
}
```

Output:

```
 Salary = 0.0
Number of workers = 100
```

Explanation:
For variable j only one block will be allocated for both the members.so first when we assigned j.salary=12.3, then memory got the value 12.3 and when we assigned j. workerNo= 100, then 100 got stored in the same memory location and 12.3 got replaced.

Difference between structure and union:



	STRUCTURE	UNION
Keyword	The keyword **struct** is used to define a structure	The keyword **union** is used to define a union.
Size	When a variable is associated with a structure, the compiler allocates the memory for each member. The size of structure is **greater than or equal to the sum of sizes of its members.**	when a variable is associated with a union, the compiler allocates the memory by considering the size of the largest memory. So, size of **union is equal to the size of largest member.**
Memory	Each member within a structure is assigned unique storage area of location.	Memory allocated is shared by individual members of union.
Value Altering	Altering the value of a member will not affect other members of the structure.	Altering the value of any of the member will alter other member values.
Accessing members	Individual member can be accessed at a time.	Only one member can be accessed at a time.
Initialization of Members	Several members of a structure can initialize at once.	Only the first member of a union can be initialized.

Program to get size of structure and union variable.

```
#include <stdio.h>
union unionJob
{
   //defining a union
   char name[32];
   float salary;
   int workerNo;
} uJob;


struct structJob
{
   char name[32];
   float salary;
   int workerNo;
} sJob;

int main()
{
   printf("size of union = %d bytes", sizeof(uJob));
   printf("\nsize of structure = %d bytes", sizeof(sJob));
   return 0;
}
```

Output:

```
size of union = 32
size of structure = 40
```

****