Structure

A structure is a user define data type used store a collection of heterogeneous type data. This data type is useful to store a record of an entity.

For example, a record of a book may contain its name, author name, ISBN number, quantity and price. Hear name author name, ISBN number are character type, whereas; quantity is an integer and price is a double type data.

- Internal data of a structure is called member of structure.
- Size of a variable of structure data type is the sum of size of its members.

Creation of a structure data type

```
struct  <Userdefined Name>
        {
        Datatype variable1;
        Datatype variable2;
        :
        };
```

Example

```
struct Student
        {
        char name[50];
        int admyear;
        double CGPA;
        };
```

Name	Admyear	CGPA
Rahul	2019	8.98

Declaration of structure variable

Creating a structure type does not occupy space in memory. To store data/ record we need to reserve space in memory. This is achieved by defining a structure type variable.

Synatx: struct <Userdefined Name> variablename;

Example: struct STUDENT student1;

	Name	Admissionyear	CGPA
student1	Joe Root	99	99.99
	2000	2050	2054

We can also declare a structure variable with creation of its structure. For this, variable names are needed to be written just after the closing curly brace and before the semi colon. In following example, s1 and s2 are two structure variable.

```
struct Student
        {
        char name[50];
        int admissionyear;
        double CGPA;
        } s1,s2;
```

Structure initialization

Structure allows direct assignment to its variable. The data type of the values must be matched with the data type of elements of structure.

```
        struct Student student1={“Rahul”, 2019, 8.98};
```

Accessing element of a structure:

A member access operator (**.**) is used to access each element.

Synatx: <structureVariable.element>.

```
        printf (“%s”,student1.name);
```

Program: An employee record includes name, code, age and salary of an employee. Input record of an employee and display all information.

```
#include<stdio.h>
struct employee
        {
        char name[30];
        char  code[10];
        int age;
        double salary;
        };
int main()
        {
        struct  employee e1;
        printf("enter the name:");
        gets(e1.name);
        printf("enter the code:");
        scanf("%s",e1.code);
        printf("Enter the age:");
        scanf("%d",&e1.age);
        printf("Enter the salary:");
        scanf("%lf",&e1.salary);
        printf("\nEmployee Name  : %20s",e1.name);
        printf("\nEmployee Code  : %20s",e1.code);
        printf("\nEmployee age   : %20d",e1.age);
        printf("\nEmployee salary: %20.2lf",e1.salary);
        return 0;
        }
```

Copying structure variable.

```
var2 = var1;
```

In this case, the content of ‘var1’ is copied to another memory location, which is identified by structure variable ‘var2’.So any change in one variable does not affect in other variable.

Pointer to structure

A structure variable is addressable. The pointer that points to the address of a structure variable called pointer to structure.

Syntax: struct <Userdefined Name> *Pointername;

```
struct STUDENT *ptr;    // ‘ptr’ is a pointer to structure.
```

Now ptr points to NULL, but can point to the structure of type STUDENT.

```
ptr = &student1;        //  ‘ptr’ points to address of structure variable ‘student1’.
```

Accessing element of a structure through its pointer

In one of following two ways; we can access the element of a structure through its pointer.

- **By using indirect member access operator (->)**

```
Syntax:    Pointername -> elementname
Example:  ptr -> name
```

- **By dereferencing the pointer and using member access operator (.)**

  ```
  Syntax:   (*Pointername).elementname
  Example:  (*ptr).name
  ```

  Note : use of parenthesis is a necessary. Writing *ptr.name causes error because the precedence of member access operator (.) is higher than that of the dereference operator (*). So compiler treats this code as *(ptr.name) .It means compiler tries to encode ptr.name first, which is itself invalid because, the operator (.) can only be used with a structure type variable, not with a pointer.

Program: Write a program to input record of two books and display the books with higher number of pages. Book record includes (bookname, authorname, price, pages)

```
#include<stdio.h>
#include<stdlib.h>
struct BOOK
     {
     char name[30];
     char  author[20];
     double price;
     int pages;
     };
int main()
     {
     struct  BOOK *ptrbook1,*ptrbook2;
     ptrbook1 = (struct BOOK *) malloc(sizeof(struct BOOK));
     printf("Enter information for First Book:\n");
     printf("Enter Book Name:");
     gets(ptrbook1->name);
     fflush(stdin);
     printf("Enter Author Name:");
     gets(ptrbook1->author);
     printf("Enter  Price:");
```

```
scanf("%lf",&ptrbook1->price);
printf("Enter the Number of pages:");
scanf("%d",&ptrbook1->pages);
ptrbook2 = (struct BOOK *) malloc(sizeof(struct BOOK));
printf("\nEnter information for Second Book:\n");
fflush(stdin);
printf("Enter Book Name:");
gets(ptrbook2->name);
fflush(stdin);
printf("Enter Author Name:");
gets(ptrbook2->author);
printf("Enter  Price:");
scanf("%lf",&ptrbook2->price);
printf("Enter the Number of pages:");
scanf("%d",&ptrbook2->pages);
if(ptrbook1->pages > ptrbook2->pages)
     {
     printf("\nBook Name  : %s",ptrbook1->name);
     printf("\nAutor Name : %s",ptrbook1->author);
     printf("\nPrice          : %0.2lf",ptrbook1->price);
     printf("\npage Counts: %20d",ptrbook1->pages);
     }
else {
     printf("\nBook Name  : %s",ptrbook2->name);
     printf("\nAutor Name : %s",ptrbook2->author);
     printf("\nPrice          : %.2lf",ptrbook2->price);
     printf("\nPage Counts: %d",ptrbook2->pages);
     }
return 0;
}
```

Nested Structure

Like any primitive type, element of a structure can be a type of another structure. This is called nested structure. So a nested structure contains another structure as one of its element. The element of inner structure is accessed by through outer structure variable.

Accessing the elements of inner structure

```
     outerstucturevariable.innerstructurevarible.elementname
```

Creating a nested structure:

```
     struct innerstructurename
          {
          datatype element;
          :
          };
     struct outerstructurename
          {
          struct innerstructurename element;
          :
          };
```

Program:. The STUDENT structure contains student name, roll number and Health data for a student. The Health data is also a structure, contains Blood group, height and weight of the student. Write a program to input all the information and displays them.

```
#include<stdio.h>
#include<stdlib.h>
struct Health  {
     char bloodgroup[3];
     float height,weight;
     };
struct Student
     {
     char name[30];
     int roll;
     struct Health hdata;
     };
```

```
int main()
     {
     struct Student s;
     printf("Enter Student Name:");
     gets(s.name);
     fflush(stdin);
     printf("Enter Roll:");
     scanf("%d",&s.roll);
     printf("Enter blood group:");
     scanf("%s",s.hdata.bloodgroup);
     printf("Enter  height(in feet):");
     scanf("%f",&s.hdata.height);
     printf("Enter  weight(in feet):");
     scanf("%f",&s.hdata.weight);
     printf("\nStdent Name  : %s",s.name);
     printf("\nRoll : %d",s.roll);
     printf("\nBG   : %s",s.hdata.bloodgroup);
     printf("\nHeight : %f",s.hdata.height);
     printf("\nWeight  : %f",s.hdata.weight);
     return 0;
     }
```

Array of structure

When an array contains multiple numbers of same structured type data as its elements, it is known as array of structure.

Creating a nested structure:

Syntax: struct <Userdefined Name> arrayname[size];

Example: struct Student students[10] ;

Program: Input records of n number of students. Print all the records.

```
#include<stdio.h>
struct employee {
     char name[30];
     char  code[10];
     int age;
     double salary;
     };
void print(struct employee [],int );
void main()
     {
     struct  employee emp[40];
     int n,i;
     printf("enter number of employees:");
     scanf("%d",&n);
     for(i=0;i<n;i++)
          {
          getchar();
          printf("\nEnter the name:");
          gets(emp[i].name);
          printf("enter the code:");
          scanf("%s",emp[i].code);
          printf("Enter the age:");
          scanf("%d",&emp[i].age);
          printf("Enter the salary:");
          scanf("%lf",&emp[i].salary);
          }
     print(emp,n);
     }
void print(struct employee emp[],int n) {
     int i;
     for(i=0;i<n;i++)
          {
     printf("\n%15s%7s\t%8d\t%0.0lf",emp[i].name,emp[i].code,emp[i].age,emp[i].salary);
          }
}
```

Program Exercise

1. Input information(Name, Roll, subject, mark) of n students. Display all the information of the student, who secured highest marks.

2. Input records (name, code, salary, age) of n employee. Sort all the records in non-decreasing order by age and then display all the records.

3. Input records (name, code, salary, address) of n employee. Delete the record of employee whose address is "Bhubaneswar".

4. Input records (name, code, salary, age) of n employee in an array. Insert another record of a new employee in a given position.

5. From records of n students(name,roll,mark), print the name of the student who secured maximum mark.

typedef keyword

typedef is use to assign a an addition name to an existing data type in order to increase the readability of the program.

syntax : typedef <existing type> < newname>;

Example : Type defining primitive types

```
typedef float fractional;
typedef int num;
num a,b,c;
factional a=2.3;
```

Example: Type defining userdefined types

```
typedef struct
    {
    char name[20];
    int age;
    }; Employee
Employee e1,e2;
```

Union

Union is a user-defined data type used to store multiple number of different type of data in one group.

It is similar to structure with respect to its definition, declaration and access method.

It is different to structure in following way----

In case of structure, internal data members take different and adjacent memory location. However, in case of union, all the members of union take the same memory location. It means same memory location is shared by all the members of union.

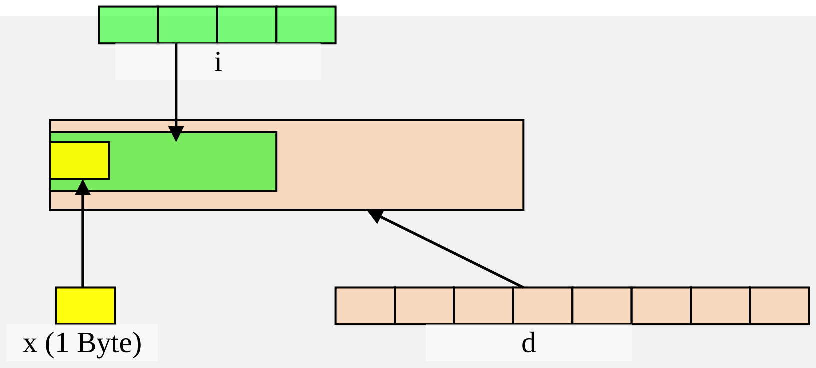
Size of a union variable is the size of the one member whose size is largest among all the members.

Creation of a union data type

```
Synatx:         union  <Userdefined Name>
                       {
                       Datatype variable1;
                       Datatype variable2;
                       :
                       };
```

Example

```
union Test
        {
        char x;
        double d
        int i;
        };
```



Declaration of union variable

Creating a structure type does not occupy space in memory. To store data/ record we need to reserve space in memory. This is achieved by defining a structure type variable.

Synatx: union <Userdefined Name> variablename;

```
Example:        union Test t;
```