Unit-2

Structures and Unions

Structure

- Structure is a user-defined datatype in C language which allows us to combine data of different types together.
- Structure is a collection of heterogeneous data elements
- Structure helps to construct a complex data type which is more meaningful.

For example: If I have to write a program to store Student information, which will have Student's name, age, branch, permanent address, father's name etc, which included string values, integer values etc,
In structure, data is stored in form of records.

Defining a structure
struct keyword is used to define a structure.
struct defines a new data type which is a collection of primary and derived datatypes.

Syntax:

```
struct structure_name
{
    //member variable 1
    //member variable 2
    //member variable 3
...
};
```

we start with the struct keyword, then it's optional to provide your structure a name, we suggest you to give it a name, then inside the curly braces, we have to mention all the member variables, which are nothing but normal C language variables of different types like int, float, array etc.

Note: The closing curly brace in the structure type declaration must be followed by a semicolon(;).

Example of Structure

Ex1:

```
struct Student
{
    char name[25];
    int age;
    char branch[10];
    char gender;
};
```

Ex2:

```
struct student
  {
    int rollno;
     char name[20];
    float percentage;
  };
```

Here struct Student declares a structure to hold the details of a student which consists of 4 data fields, namely name, age, branch and gender. These fields are called **structure elements or members**.

Each member can have different datatype, like in this case, name is an array of char type and age is of int type etc. **Student** is the name of the structure and is called as the **structure tag**.

Note: Structure members can not be initialized during structure definition time.

Declaring Structure Variables

- It is possible to declare variables of a **structure**, either along with structure definition or after the structure is defined.
- **Structure** variable declaration is similar to the declaration of any normal variable of any other datatype.

Structure variables can be declared in following two ways:

1.Declaring Structure variables separately

2. Declaring Structure variables with structure definition

1) Declaring Structure variables separately

Syntax: struct structurename var1,var2…;

```
struct Student
{
   int rollno;
   char name[25];
   float percentage;
};
struct Student S1, S2;      //declaring variables of struct Student
```

2) Declaring Structure variables with structure definition

```
struct Student
{
   int rollno;
   char name[25];
   float percentage;
}S1, S2;
```

Here S1 and S2 are variables of structure Student. However this approach is not much recommended.

Accessing Structure Members

Structure members can be accessed with structure operators along with structure variables.

2 structure operators

1. (.) dot operator
2. (->) arrow operator

Structure member access operators:

Dot(.) operator: this operator can be used to access structure members with structure variables.

Syntax: structurevariable.structuremember

struct student s;

Ex: s.rollno, s.name, s.percentage

Arrow(->) operator: this operator can be used to access structure members with structure pointer variables.

Syntax: structurepointervariable->structuremember

struct student *p;

Ex: p->rollno, p->name, p->percentage

- Structure members can be accessed and assigned values in a number of ways. Structure members have no meaning individually without the structure variables.

For example:

```
//Using dot operator
#include<stdio.h>
#include<string.h>
struct Student
{
   int rollno;
   char name[25];
   float percentage;
};
void main()
{
   struct Student s;
    s.rollno = 18;
    s.percentage = 93.23;
    strcpy(s.name, "sita");
    printf("Rollno of Student 1: %d\n", s.rollno);
    printf("Name of Student 1: %s\n", s.name);
   printf("percentage of student 1: %f\n", s.percentage);
 }
```

```
//Using Arrow operator
#include<stdio.h>
#include<string.h>
struct Student
{
   int rollno;
   char name[25];
   float percentage;
};
void main()
{
   struct Student *p;
   p=(struct Student *)malloc(sizeof(struct Student));
    p->rollno = 18;
    p->percentage = 93.23;
    strcpy(p->name, "sita");
    printf("Rollno of Student 1: %d\n", p->rollno);
    printf("Name of Student 1: %s\n", p->name);
   printf("percentage of student 1: %f\n", p->percentage);
 }
```

Memory Allocation for Structure

- Memory can be allocated for structure variable not for structure definition

- When we create a structure variable, then compiler allocate contiguous memory for the data members of the structure.
- The size of allocated memory is the sum of sizes of all data members.
- The data members of structure is accessed with the help of the structure variable and structure member name.

Ex:

```
struct Student
{
   int rollno;
   char name[25];
   float percentage;
};
Struct student s;
```

Note: Memory can not be created for student. Memory can be created for structure student variable s.

```
rollno– 2 bytes, name- 20 bytes , percentage – 4 bytes
so total memory allocated for s is 26 bytes(2+20+4).
```

Structure Initialization at compile time

Like a variable of any other datatype, structure variable can also be initialized at compile time.

```
struct Student
{
   int rollno;
   char name[25];
   float percentage;
};
struct Student s = { 18 , “sita”, 93.23 };    //initialization
```

or,

```
struct Student s;
s.rollno = 18;     //initialization of each member separately
s.name = “sita”;
s.percentage = 93.23;
```

Structure Initialization at Run time

```
//Read and display student details
#include<stdio.h>
#include<string.h>
struct Student
{
   int rollno;
   char name[25];
   float percentage;
};
void main()
{
   struct Student s;
   printf(“Enter rollno, name and percentage\n”);
    scanf(“%d”,&s.rollno );
   gets(s.name);
    scanf(“%f”,&s.percentage);
     printf("Rollno of Student : %d\n", s.rollno);
    printf("Name of Student : %s\n", s.name);
```

```
   printf("percentage of student : %f\n", s.percentage);
 }
```

```
//Read and display student details using structure pointer
#include<stdio.h>
#include<string.h>
struct Student
{
   int rollno;
   char name[25];
   float percentage;
};
void main()
{
   struct Student *p;
   p=(struct student *)malloc(sizeof(struct student));
   printf(“Enter rollno, name and percentage\n”);
    scanf(“%d”,&p->rollno );
   gets(p->name);
    scanf(“%f”,&p->percentage);
     printf("Rollno of Student : %d\n", p->rollno);
    printf("Name of Student : %s\n", p->name);
   printf("percentage of student : %f\n", p->percentage);
 }
```

```
//Read and display student details
#include<stdio.h>
#include<string.h>
struct employee
{
   char    name[30];
   int     empId;
   float   salary;
};
void main()
{
   struct employee emp;
    printf("\nEnter details :\n");
```

```
   printf("Name :");
  gets(emp.name);
   printf("ID ?:");
   scanf("%d",&emp.empId);
   printf("Salary ?:");
   scanf("%f",&emp.salary);
    printf("\nEntered detail is:");
   printf("Name: %s"    ,emp.name);
   printf("Id: %d"      ,emp.empId);
   printf("Salary: %f\n",emp.salary);
}
```

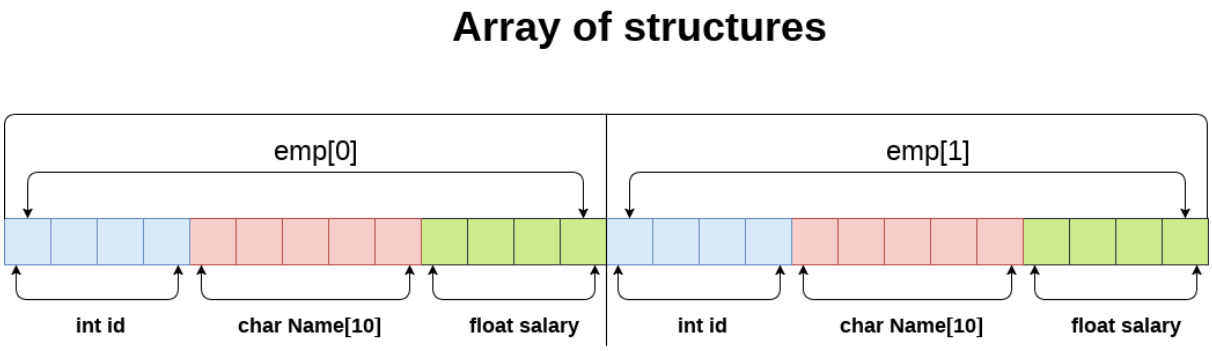
```
//addition of 2 complex numbers using structures
#include <stdio.h>
struct complex
{
   float real;
   float imag;
};
void main()
{
   struct complex c1, c2, c3;
   printf("Enter c1 real and imag\n");
   scanf("%f%f", &c1.real, &c1.imag);
   printf("Enter c2 real and imag\n");
   scanf("%f%f", &c2.real, &c2.imag);
   c3.real = c1.real + c2.real;
   c3.imag = c1.imag + c2.imag;
   printf("Sum = %f + i %f", c3.real, c3.imag);

}
```

```
//multiplication of 2 complex numbers using structures
#include <stdio.h>
struct complex
{
   float real;
   float imag;
};

void main()
{
   struct complex c1, c2, c3;

   printf("Enter C1 the real and imaginary parts: ");
   scanf("%f%f", &c1.real, &c1.imag);
   printf("Enter C2 the real and imaginary parts: \n");
```

```
    scanf("%f%f", &c2.real, &c2.imag);
    c3.real = (c1.real*c2.real)-(c1.img*c2.imag);
    c3.imag = (c1.real * c2.imag)+(c2.real*c1.imag);
     printf("multiplication = %f + i %f", c3.real, c3.imag);

}
```

Array of structures

- An array of structures in C can be defined as the collection of multiple structures variables where each variable contains information about different entities.
- The array of structures in C are used to store information about multiple entities of different data types.
- The array of structures is also known as the collection of structures.




```
struct employee
{
    int id;
    char name[5];
    float salary;
};
struct employee emp[2];
```

sizeof (emp) = 4 + 5 + 4 = 13 bytes

sizeof (emp[2]) = 26 bytes

//Read and display array of students details

```
#include<stdio.h>
#include <string.h>
struct student
{
    int rollno;
    char name[10];
    float percentage;
};
void main()
{
    int i,n;
    struct student s[10];
    printf("Enter  no of Records\n");
    scanf("%d",&n);
```

```
    printf("Enter Records %d of  students\n",n);
   for(i=0;i<n;i++)
   {
       printf("\nEnter Rollno:");
       scanf("%d",&s[i].rollno);
       printf("\nEnter Name:");
       __fpurge(stdin);
       gets(s[i].name);
       printf("\nEnter percentage:");
       scanf("%f",&s[i].percentage);
    }
    printf("\nStudent Information List:");
    for(i=0;i<n;i++)
    {
         printf("\nRollno:%d, Name:%s,Percentage:%f\n",s[i].rollno,s[i].name,s[i].percentage);
    }

}
```

OUTPUT:

```
Enter  no of Records
5
Enter Records of 5 students
Enter Rollno:1
Enter Name:Sonoo
Enter Rollno:2
Enter Name:Ratan
Enter Rollno:3
Enter Name:Vimal
Enter Rollno:4
Enter Name:James
Enter Rollno:5
Enter Name:Sarfraz

Student Information List:
Rollno:1, Name:Sonoo
Rollno:2, Name:Ratan
Rollno:3, Name:Vimal
Rollno:4, Name:James
Rollno:5, Name:Sarfraz
```

// Read and display all employee details in organization

```
#include<stdio.h>
#include<string.h>
struct employee
{
   int id,salary;
   char name[20],desg[20];
};
 void main()
{
   int i,n;
   struct employee emp[10];
   printf("Enter the no of employees\n");
   scanf("%d",&n);
   printf("Enter employee details \n");
   for(i=0;i<n;i++)
   {
      printf("Enter employee details as id , name , designation , salary\n");
      scanf("%d%s%s%d",&emp[i].id, emp[i].name, &emp[i].desg, &emp[i].salary);
   }
   printf("Enter employee details are\n");
   for(i=0;i<n;i++)
   {
      printf("id:%d,name:%s,designation:%s,salary:%d\n",&emp[i].id, emp[i].name,
&emp[i].desg, &emp[i].salary);
   }

}
```

Structure member as an Array or Arrays within structure

Sometimes, arrays may be the member within structure, this is known as arrays within structure. Accessing arrays within structure is similar to accessing other members.

```
struct student
{
   int rollno;
   char name[20];
    int marks[5];      //array as a member of structre
   float percentage;
};
```

If s is a variable of type struct student then:

s.marks[0] – refers to the marks in the first subject
s.marks[1] – refers to the marks in the second subject

//Array within structure

```
struct student
{
   char name[20];
   int rollno;
   int marks[5];
   float percentage;
};
 void main()
{
     int i,sum=0;
     struct student s;
     printf("\nEnter Rollno:");
     scanf("%d",&s.rollno);
     printf("\nEnter Name:");
     gets(s.name);
     printf("\nEnter 5 subjects marks\n:");
      for(i=0;i<5;i++)
     {
          scanf("%d",&s.marks[i]);
          sum=sum+s.marks[i];
      }
      s.percentage=(float)sum/5;
     printf("\n student details\n");
     printf("\nRollno:%d, Name:%s",s.rollno,s.name);
     printf("Student Marks\n");
      for(i=0;i<5;i++)
     {
          printf("%d\t",s.marks[i]);
      }
      Printf("percentage=%f\n",s.percentage);
}
```

//Array of structures and Array within structure

```
struct student
{
   char name[20];
   int rollno;
   int marks[5];
   float percentage;
};
 void main()
{
     int i,j,sum=0,n;
     struct student s[10];
     printf("Enter no of students\n");
     scanf("%d",&n);
     for(i=0;i<n;i++)
     {
         sum=0;
         Printf("Enter %d index student details\n",i);
```

```
        printf("\nEnter Rollno:");
        scanf("%d",&s[i].rollno);
        printf("\nEnter Name:");
         __fpurge(stdin);
        gets(s[i].name);
        printf("\nEnter 5 subjects marks\n:");
        for(j=0;j<5;j++)
        {
            scanf("%d",&s[i].marks[j]);
            sum=sum+s[i].marks[j];
        }
     s[i].percentage=(float)sum/5;
  }
    printf("\n student details\n");
   printf("Rollno\tName\t  marks\tpercentage\n");
   for(i=0;i<n;i++)
   {
      printf("%d\t%s\t",s[i].rollno,s[i].name);
       for(j=0;j<5;j++)
       {
           printf("%d ",s[i].marks[j]);
       }
     printf("%f\n",s[i].percentage);
  }
}
```

Nested Structures / Structure with in structure

- Nested structure in C is nothing but structure within structure. One structure variable can be declared inside other structure as a structure member.
- The structure variables can be a normal structure variable or a pointer variable to access the data.
- When a structure contains another structure, it is called nested structure.

1. Structure within structure in C using normal variable
2. Structure within structure in C using pointer variable

- **Structure within structure in C using normal variable**

Syntax

```
struct structure1
{
    ----------
    ----------
};
struct structure2
{
    ----------
    ----------
    struct structure1 X;
    ----------------
};
//structure2 is nested structure
```

```
struct structure2
{
    ----------
    ----------
    struct structure1
    {
        ----------
        ----------
    }X;
    ----------------
};

//structure2 is nested structure
```

Example:
struct Date
{
int day;
int month;
int year;
};
struct student
{
int rollno;
char name[20];
struct Date d;
float percentage;
};
	Example:
	struct student
	{
	int rollno;
	char name[20];
	struct Date
	{
	int day;
	int month;
	int year;
	}d;
	float percentage;
	};

- **Structure within structure in C using pointer variable**

Syntax

struct structure1 { - }; struct structure2 { - **struct structure1 *X;** ---------- };	struct structure2 { - **struct structure1** **{** - **}*X;** ---------- };

Example:
struct Date
{
int day;
int month;
int year;
};
struct student
{
int rollno;
char name[20];
struct Date *d;
float percentage;
};
	Example:
	struct student
	{
	int rollno;
	char name[20];
	struct Date
	{
	int day;
	int month;
	int year;
	}*d;
	float percentage;
	};

Program:Program to read and display student structure details using nested structure as structure variable

Hint: Use date of birth for nested structure.

```
#include<stdio.h>

#include<string.h>

struct Date
{
   int day;
   int month;
   int year;
};
struct student
{
     int rollno;
     char name[20];
     struct Date d;
     float percentage;
};

void main()
{
    struct Student s;
    printf(“Enter rollno, name and percentage\n”);
    scanf(“%d”,&s.rollno );
    gets(s.name);
    scanf(“%f”,&s.percentage);
    printf(“Enter Date of birth day,month and year\n”);
    scanf(“%d%d%d”,&s.d.day,&s.d.month,&s.d.year);
    printf("Rollno of Student : %d\n", s.rollno);
    printf("Name of Student : %s\n", s.name);
    printf(“DOB : %d-%d-%d”,s.d.day,s.d.month,s.d.year);
    printf("percentage of student : %f\n", s.percentage);
 }
```

Program:Program to read and display student structure details using nested structure as structure pointer variable

Hint: Use date of birth for nested structure.

```
#include<stdio.h>

#include<string.h>

struct Date
{
   int day;
   int month;
   int year;
};

struct student
{
     int rollno;
     char name[20];
     struct Date *d;
     float percentage;
};

void main()
{
    struct Student s;
    printf("Enter rollno, name and percentage\n");
    scanf("%d",&s.rollno );
    gets(s.name);
    scanf("%f",&s.percentage);
    printf("Enter Date of birth day,month and year\n");
    scanf("%d%d%d",&s.d->day,&s.d->month,&s.d->year);
    printf("Rollno of Student : %d\n", s.rollno);
    printf("Name of Student : %s\n", s.name);
    printf("DOB : %d-%d-%d",s.d->day,s.d->month,s.d->year);
    printf("percentage of student : %f\n", s.percentage);
 }
```

//Array of Nested structures

```
struct Date
{
   int day;
   int month;
   int year;
};
struct student
{
     int rollno;
     char name[20];
     struct Date d;
     float percentage;
};
```

```
void main()
{
    struct Student s[10];
    int i,n;
    printf(“Enter no of students\n”);
    scanf(“%d”,&n);
   printf(“Enter details\n”);
   for(i=0;i<n;i++)
   {
       printf(“Enter rollno, name and percentage\n”);
       scanf(“%d”,&s[i].rollno );
       gets(s[i].name);
       scanf(“%f”,&s[i].percentage);
       printf(“Enter Date of birth day,month and year\n”);
      scanf(“%d%d%d”,&s[i].d.day,&s[i].d.month,&s[i].d.year);
  }
  for(i=0;i<n;i++)
   {
    printf("Rollno of Student : %d\n", s[i].rollno);
    printf("Name of Student : %s\n", s[i].name);
    printf(“DOB : %d-%d-%d”,s[i].d.day,s[i].d.month,s[i].d.year);
    printf("percentage of student : %f\n", s[i].percentage);
  }
 }
```

Self-Referential structures

Self- Referential structures are those structures that have one or more pointers which point to the same type of structure, as their member.

Note: Structure member may be pointer variable of same structure but not Normal variable.

Ex1:	Ex2:

struct node
{
int data;
struct node *next;
};
node is self-referential structure
	struct student
	{
	int rollno;
	char name[20];
	float percentage;
	struct student *p;
	};
	Student is self-referential structure

Structures and Functions

Like all other types, we can pass structures as arguments to a function. In fact, we can pass, individual members, structure variables, a pointer to structures etc to the function. Similarly, functions can return either an individual member or structures variable or pointer to the structure.

We can pass the C structures to functions in 3 ways:

1. Passing each item of the structure as a function argument. It is similar to passing normal values as arguments. Although it is easy to implement, we don't use this approach because if the size of a structure is a bit larger, then our life becomes miserable.
2. Pass the whole structure as a value.
3. We can also Pass the address of the structure (pass by reference).

➢ Passing Structure Members as arguments to Function

We can pass individual members to a function just like ordinary variables.

Example: A program demonstrates how to pass structure members as arguments to the function.

```
#include<stdio.h>
#include<string.h>
struct student
{
   int rollno;
   char name[20];
   float percentage;
};

 void display(int ,char [] , float);          //Function declaration

void main()
{
   struct student s = {101,"sonoo",78.9};
   display(s.rollno, s.name,  s.percentage);     //passing structure members as parameters
}

void display(int rollno, char name[],  int percentage)
{
     printf("Roll no: %d\n", rollno);
     printf("Name: %s\n", name);
     printf("Percentage: %f\n", percentage);
}
```

➢ Passing Structure Variable as Argument to a Function

- If a structure contains two-three members then we can easily pass them to function.
- But if a structure contains more number of members but what if members is an error-prone process. So in such cases instead of passing members individually, we can pass structure variable itself.

Example: program demonstrates how we can pass structure variable as an argument to the function.

```
#include<stdio.h>
#include<string.h>
struct student
{
   int rollno;
   char name[20];
   float percentage;
};

void display(struct student );

void main()
{
   struct student s = {10, "George", 69};
   display(s);
}

void display(struct student p)
{
   printf("Roll no: %d\n", p.rollno);
   printf("Name: %s\n", p.name);
   printf(“Percentage: %f\n", p.percentage);
}
```

- **Passing Structure Pointers as Argument to a Function**

Although passing structure variable as an argument allows us to pass all the members of the structure to a function there are some downsides to this operation.

1. Recall that a copy of the structure is passed to the formal argument. If the structure is large and you are passing structure variables frequently then it can take quite a bit of time which make the program inefficient.
2. Additional memory is required to save every copy of the structure. Memory can be saved when we pass structure pointer to a function instead of passing structure variable.

Example: program demonstrates how to pass structure pointers as arguments to a function.

```
#include<stdio.h>
#include<string.h>
struct student
{
   int rollno;
   char name[20];
   float percentage;
};

void display(struct student *);
```

```
void main()
{
   struct student s= {25,"Jane", 78.9};
   display(&s);
}

void display(struct student *p)
{
   printf("Roll no: %d\n",p->rollno);
   printf("Name: %s\n", p->name);
   printf("Percentage: %d\n", p->percentage);}
}
```

User Defined Datatypes

User Defined Datatypes are the new datatypes created by the user.

There are 2 types of user defined datatypes

1. **typedef (type definition)**
2. **enum (enumerated datatype)**

typedef

- typedef is a keyword and it is a statement that is used to create a new user defined datatypes.
- This is same like defining alias for the commands.

Syntax: typedef datatype newname;

- typedef: It is a keyword.
- datatype: It is the name of any existing type or user defined type created using structure/union.
- newname: alias or new name you want to give to any existing type or user defined type.

Example:

```
typedef int length;      //length is a new datatype name , it is not variable.

length x,y;              // x and y are the variables of type length.
```

#program to illustrate typedef

```
#include<stdio.h>
typedef int length;
void main()
{
    length x;
   printf("Enter length value x\n");
    scanf("%d",&x);
    printf("x=%d\n", x);
}
```

Creating a structure using typedef statement

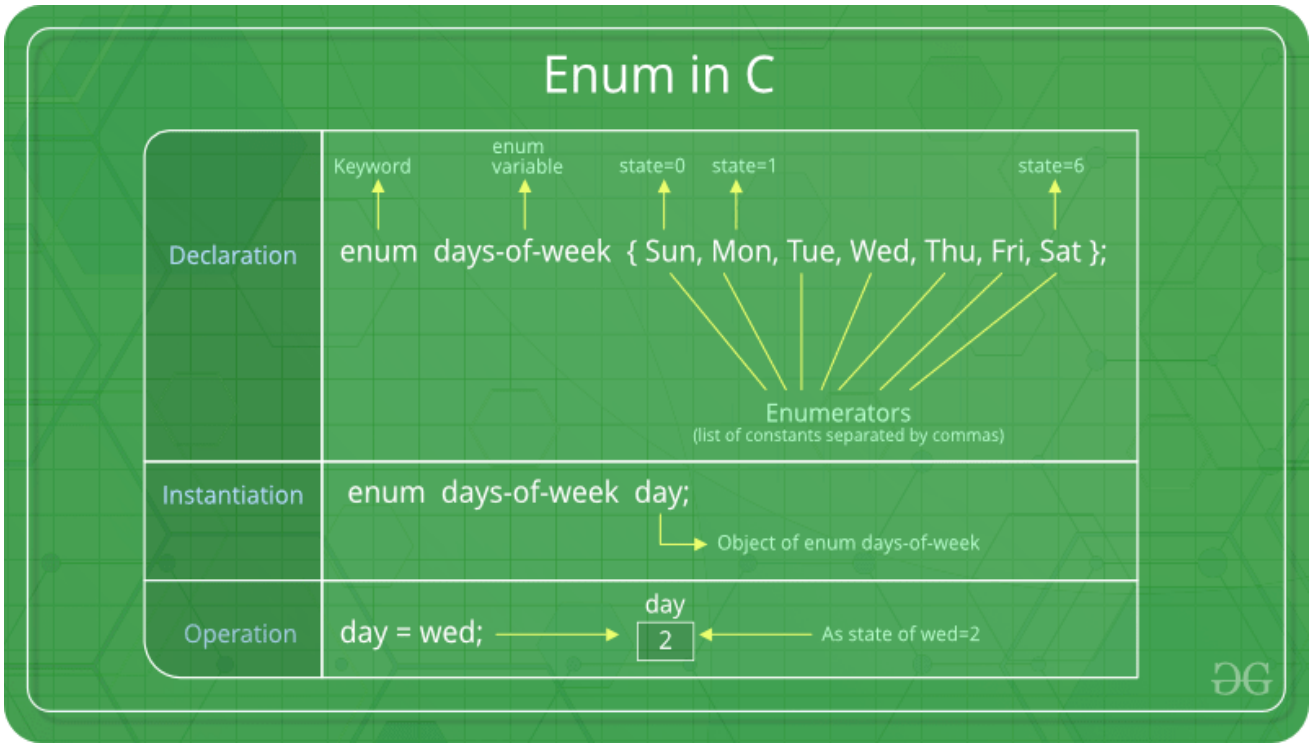
Syntax	**Example**
typedef struct <structure_name> { --------- --------- --------- }Newdatatype_name; • Here structure_name only optional to mention. We can avoid structure names when we are defining structures using typedef. • Structure name is not datatype. • To declare structure variable struct keyword is not required. **Syntax:** **newdatatypename var;**	typedef struct VJIT { int rollno; char name[20]; float percentage; struct student *p; }student; • VJIT is a structure name only • Student is a new datatype name And it can be used to create variables without using struct keyword. **Example:** **student s;**

enum: Enumerated datatype

Enumeration (or enum) is a user defined data type in C. It is mainly used to assign names to integral constants, the names make a program easy to read and maintain.

Syntax:

enum newdataypename{value-1,value-2…….value-n};



Ex : enum week{Mon, Tue, Wed, Thu, Fri, Sat, Sun}; //week is enumerated datatype

week d; //d is a variable of type week

d= wed; ie d=2

Program to illustrate enum

```
#include<stdio.h>
 enum week{Mon, Tue, Wed, Thu, Fri, Sat, Sun};

void main()
{
   enum week d;
   d = Wed;
   printf("d=%d",d);
}
```

Output: d= 2

Unions

- A union is a special data type available in C that allows to store different data types in the same memory location.
- You can define a union with many members, but only one member can contain a value at any given time.
- Unions provide an efficient way of using the same memory location for multiple-purpose.
- **The memory occupied by a union will be the sizeof the memory of a larger union member.**

Defining a Union

- To define a union, you must use the **union** keyword in the same way as you did while defining a structure.
- The union statement defines a new data type with more than one member.

Syntax	**Example**
union unionname { member definition; member definition; ... member definition; } ;	union Data { int i; float f; char str[20]; } ;
Declaring union variable union unionname var1,var2;	union Data d;

In the above example

- Now, a variable of **Data** can store an integer, a floating-point number, or a string of characters.
- It means a single variable, i.e., same memory location, can be used to store multiple types of data.
- You can use any built-in or user defined data types inside a union based on your requirement.

Accessing Union Members

- To access any member of a union, we use the member access operator dot (.).
- The member access operator is coded as a period between the union variable name and the union member that we wish to access.
- You would use the keyword union to define variables of union type.

Example

```
#include <stdio.h>
#include <string.h>
 union Data
 {
    int i;
    float f;
    char str[20];
};

void main( )
{
  union Data d;
  d.i = 10;
  d.f = 220.5;
  strcpy( d.str, "C Programming");
  printf( "d.i : %d\n", d.i);
  printf( "d.f : %f\n", d.f);
  printf( "d.str : %s\n", d.str);
}
```

OUTPUT
d.i : 1917853763
d.f : 412236058032779486045275999436 8.000000
d.str : C Programming

Here, we can see that the values of i and f members of union got corrupted because the final value assigned to the variable has occupied the memory location and this is the reason that the value of str member is getting printed very well.

```
void main( )
{
  union Data d;

  d.i = 10;
  printf( "d.i : %d\n", d.i);
  d.f = 220.5;
  printf( "d.f : %f\n", d.f);
  strcpy( d.str, "C Programming");
  printf( "d.str : %s\n", d.str);
}
```

OUTPUT
d.i : 10
d.f : 220.5
d.str : C Programming

Differences between structures and Unions

	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.