UNIT - 1: Structures, Union and Files

Outline:

- **Structures**
- **Union**
- **Bitfields**
- **Command Line Arguments**
- **Files**

Structures

Outline:

- **Definition**
- **Defining a Structure**
- **Declaring a Structure Variable**
- **Pointer to a Structure Variable**
- **Accessing Structure Members**
- **Structure Initialization**
- **Memory Map for a Structure**
- **Structure Operations**
- **Array within a Structure**
- **Nested Structure**
- **Array of Structures**
- **Structures and Functions**

Structure

- Structure is an user-defined data type in C.

Definition:

A structure is an ordered collection of logically related data items under a single name.

Defining a structure/Structure Definition:

Syntax:

```
struct tagname
{
        datatype member1;
        datatype member2;
        . . .
};
```

where,

struct is a keyword in C which tells the compiler that structure has been defined
tagname is the name of the structure
datatype can be int, float, char etc.
member1, member2,... are called the members/fields/elements of the structure

Example1:

```
struct Student
{
        char name[20];
        int roll;
        float percent;
        char branch[20];
};
```

Here,

Student is the name of the structure.
name, roll, percent and branch are the members.

Example2:

```
struct Book
{
        char title[30];
        char author[30];
        int pages;
        float price;
};
```

Here,

Book is the name of the structure.
title, author, pages and price are the members.

Declaring a structure variable

Method 1: Structure Definition and variable declaration can be written as two separate statements.

Syntax:

```
struct tagname var1,var2,...;
```

Examples:

```
struct Student s1,s2; //Here, s1 and s2 are variable of struct Student type
struct Book b;        //Here, b is a variable of struct Book type
```

Method 2: Structure Definition and variable declaration can be combined and written as a single statement.

Syntax:

```
struct tagname
   {
        datatype member1;
        datatype member2;
         . . .
    }var1,var2,...;
```

Note: Specifying tagname is optional.

Example:

```
struct Student
{
        char name[20];
        int roll;
        float percent;
}s1,s2;
```

Here,

Student is the name of the structure.
name, roll, percent and branch are the members.
s1 and s2 are called structure variables of struct Student type.

Method 3: Typedefined structure

Syntax for structure definition:

```
typedef struct tagname
   {
          datatype member1;
          datatype member2;
           . . .
    }type_ID;
```

Syntax for variable declaration:

```
type_ID var1,var2,...;
```

Example1:

```
typedef struct
{
     char name[20];
     int roll;
     float percent;
     char branch[20];
 }Student;
 Student s1,s2;
```

Here, s1 and s2 are called structure variables.

Example2:

```
typedef struct book
{
      char title[30];
      char author[30];
      int pages;
      float price;
}Book;
Book b1,b2,b3;
struct book b4;
```

Here, b1, b2, b3 and b4 are called structure variables.

Pointer to a structure variable

A variable that holds the address of a structure variable is referred to as a pointer to a structure variable.

Declaration Syntax:

```
struct tagname *ptr_var;
```

Initialization Syntax:

```
ptr_var = &struct_var;
```

Examples:

```
struct Student s1,s2,*p;
struct Book b,*q;
```

Examples:

```
p=&s2;
q=&b;
```

Example:

```
struct Student
{
      char name[20];
      int roll;
      float percent;
};
struct Student s,*p=&s;
```

Accessing structure members:

- Structure members can be accessed only through structure variable.

- **Three ways to access structure members:**
  - Using dot operator (for normal structure variables)
  - Using indirection operator
  - Using selection operator

  } (for pointer to structure variable)

Syntax for accessing Structure member:

Using dot operator:	struct_variable.struct_member
Using indirection operator:	(*ptr_var).struct_member
Using selection operator:	ptr_var -> struct_member

Example:	dot operator	indirection operator	selection operator
struct Student	s.name	(*p).name	p->name
{	s.roll	(*p).roll	p->roll
char name[20];	s.percent	(*p).percent	p->percent
int roll;			
float percent;			
}s,*p=&s;			

Structure Initialization:

- Assigning values to the members is referred to as initialization.
- A structure variable can be initialized either during compile time or during run-time.

Compile-time initialization:

Initializing a structure variable during compile time is referred to as compile-time initialization.

Syntax:

```
struct tagname
{
        datatype member1;
        datatype member2;
        . . .
};
struct tagname var = {val1,val2,...};
```

Example:

```
struct Student
{
        char name[20];
        int roll;
        float percent;
};
struct Student s = {“abc”,15,95.0};
```

Rules for Compile-time Initialization:

1. We cannot initialize the individual structure members within the structure definition.
2. The order in which values are specified within the braces must match the order of the members within the structure definition.
3. Partial initialization is allowed i.e., we can initialize the first few members and leave the rest uninitialized.
   - Uninitialized members take the default value as follows:
     - Zero for int or float type
     - NULL for char or string

Example:

```
int main()
{
      struct Student
      {
            char name[20];
            int roll;
            float percent;
      };
      struct Student s = {“abc”,15};
      printf(“\nName: %s\nRoll No.: %d\nPercentage: %f”,s.name,s.roll,s.percent);
      return 0;
}
```

Run-time initialization:

Initializing a structure variable during execution time/run-time is referred to as run-time initialization.

Example:

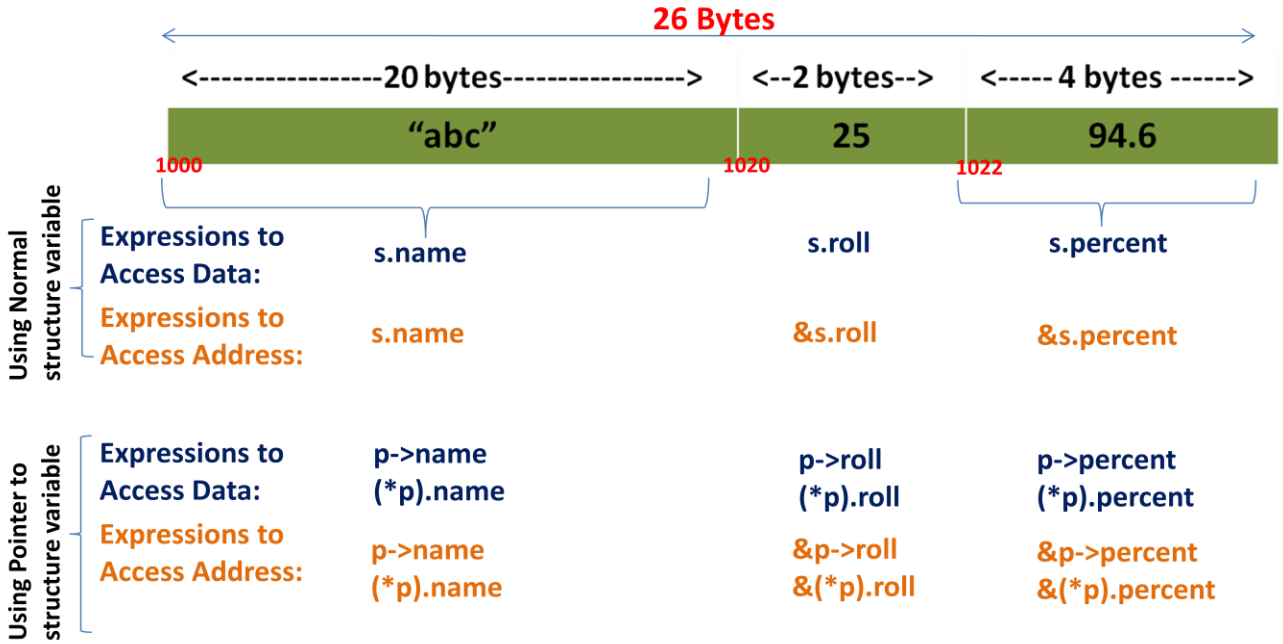
```
int main()
{
       struct Student
      {
           char name[20];
           int roll;
           float percent;
      };
```

```
    struct Student s ;
    printf("\nEnter name, roll no. and percentage of a student:\n");
    scanf("%s%d%f",s.name,&s.roll,&s.percent);
    printf("\nName:  %s\nRoll No.: %d\nPercentage: %f",s.name,s.roll,s.percent);
    return 0;
}
```

Memory map for a Structure

Example: Consider the following structure definition and declaration:

```
struct Student
{
        char name[20];
        int roll;
        float percent;
}s={"abc",25,94.6},*p=&s;
```



To find the address of each member, given with Base address as 1000:

- Address of s.name = Address of s = 1000
- Address of s.roll = Base address + sizeof(s.name) = 1000 + 20 = 1020
- Address of s.percent = Base address + sizeof(s.name) + sizeof(s.roll)

  = 1000 + 20 + 2 = 1022

Example C Program to read and print Student information using normal structure variable

```
#include<stdio.h>
int main()
{
	struct Student
	{
		char name[20];
		 int roll;
		 float percent;
	};
	struct Student s;
	printf("\nEnter the name, roll no. and percentage of a student:\n");
	scanf("%s%d%f",s.name,&s.roll,&s.percent);
	printf("\nName: %s\nRoll No.: %d\nPercentage: %f",s.name,s.roll, s.percent ) ;
	return 0;
}
```

Example C Program to read and print Student information using pointer to a structure variable

```
#include<stdio.h>
int main()
{
	struct Student
	{
		char name[20];
		 int roll;
		 float percent;
	};
	struct Student s,*p=&s;
	printf("\nEnter the name, roll no. and percentage of a student:\n");
	scanf("%s%d%f",p->name,&p->roll,&p->percent);
	printf("\nName: %s\nRoll No.: %d\nPercentage: %f",p->name,p->roll,p->percent);
	return 0;
}
```

Operations on Structure

Contents of one structure variable can be copied to another variable if both the variables are of same structure type.

Example: Consider the following structure

```
struct Data
{
        char ch;
        int a;
        float b;
        char str[20];
};
struct Data d1={'A',12,3.5,"xyz"},d2;
d2 = d1;
```

Address of a structure variable can be accessed using & operator.

Example1: Consider the following structure

```
int main()
{
        struct Data
       {
              char ch;
              int a;
              float b;
              char str[20];
       };
       struct Data d1,*p=&d1;

       printf("\nBase address =%u",&d1);
       printf("\nBase address =%u",&d1.ch);
       printf("\nBase address =%u",p);

       return 0;
}
```

Two structure variables cannot be compared whether they are of same structure type or of different type.

We can perform arithmetic, logical, relational, bitwise operations etc. on individual structure members.

Example1: Consider the following structure

```
struct Data
{
        int a;
        float b;
};
struct Data d1={15,4.5},d2={10,3.2};
```

Following expressions are valid:

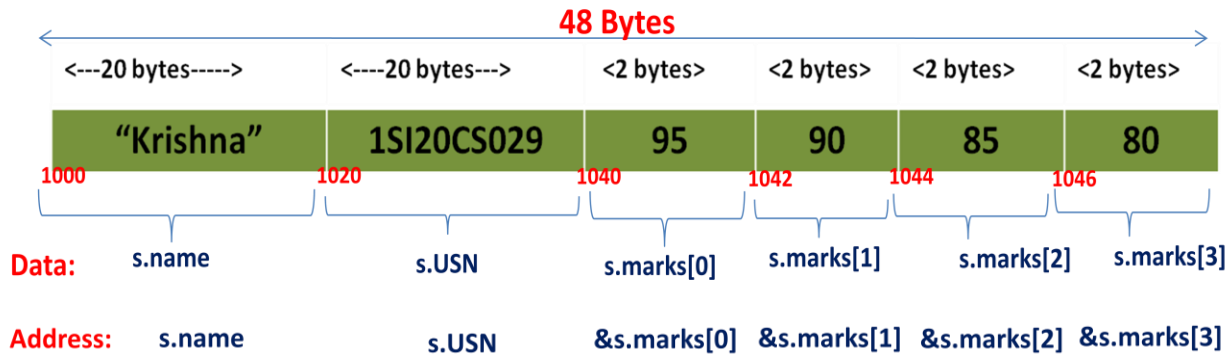
```
d1.a=d1.a + 10;
d1.a++;
d1.a==d2.a;
d1.a < d2.a;
d1.a = d1.a <<2;
```

Array within a Structure

We can have array as structure members.

Example: Consider the following structure

```
struct Student
{
        char name[20];
        char USN[20];
        int marks[4];
};
struct Student s={"Krishna",1SI20CS029,95,90,85,80},*p=&s;
```



Example1: C program to compute the sum of marks scored by a student.

```
int main()
{
      struct Student
      {
            char name[20];
            char USN[20];
            int marks[4];
      };
      struct Student s;
      int i,total=0;
      printf("\nEnter name and USN of the student:\n");
      scanf("%s%s",s.name,s.USN);
      printf("\nEnter the marks scored in 4 subjects:\n");
      for(i=0;i<4;i++)
      {
            scanf("%d",&s.marks[i]);
            total = total  +  s.marks[i];
      }

      printf("\nStudent Details.....\n");
      printf("Name: %s\nUSN:%s\nTotal: %d",s.name,s.USN,total);
      return 0;
}
```

Example2: C program to compute the sum of all elements in an array using structure.

```
int main()
{
     struct Array
    {
         int a[20];
         int n;
   };

    struct Array p;
    int i,sum=0;

    printf("\nEnter the size of the array: ");
    scanf("%d",&p.n);
```

```
    printf("\nEnter %d elements for the array:\n",p.n);

    for(i=0;i<p.n;i++)
    {
          scanf("%d",&p.a[i]);
         sum=sum +p.a[i];
     }
     printf("\nSum of all elements in the array = %d",sum);
     return 0;
}
```

Nested Structure/Embedded Structure

- A structure within a structure or a structure containing a member of another structure type is referred to as a Nested Structure.

- **Syntax1:**

```
struct tagname1
{
        datatype member1a;
        datatype member1b;
        . . .
        struct tagname2
       {
              datatype member2a;
              datatype member2b;
              . . .
       }var2a,var2b,...;
};
struct tagname1 var1a,var1b,...;
```

- Innermost member in a nested structure can be accessed by *chaining all corresponding structure variables with structure member* using dot operator.

- **Example:**

```
struct Employee
{
      char name[20];
      int id;
```

```
    char dept[20];
    struct Salary
   {
         int basic;
         int da;
         int hra;
    }s;
};
struct Employee e;
```

- **Syntax2:**

```
struct tagname2
{
      datatype member2a;
      datatype member2b;
       . . .
};
 struct tagname1
 {
      datatype member1a;
      datatype member1b;
       . . .
       struct tagname2 var2a,var2b,...;
 };
 struct tagname1 var1a,var1b,...;
```

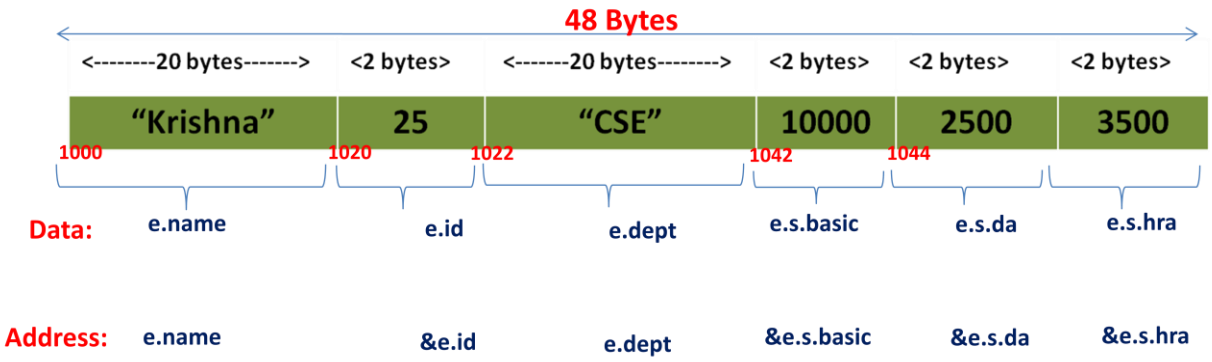
- **Example:**

```
 struct Salary
{
      int basic;
      int da;
      int hra;
 };
 struct Employee
{
      char name[20];
      int id;
      char dept[20];
      struct Salary s;
};
struct Employee e;
```

Memory map for a Nested Structure

Example: Consider the following nested structure

```
struct Salary
{
        int basic,da,hra;
};
struct Employee
{
        char name[20];
        int id;
        char dept[20];
        struct Salary s;
}e={"Krishna",25,"CSE",10000,2500,3500},*p=&e;
```



Example1: C Program to compute gross salary of the employee using nested structure.

```
int main()
{
        struct Salary
        {
              int basic,da,hra;
        };
        struct Employee
        {
                    char name[20];
                    int id;
                    char dept[20];
                    struct Salary s;
           };
           struct Employee e;
```

```
        int gross;
        printf("\nEnter the name, id and dept of the employee:\n");
        scanf("%s%d%s",e.name,&e.id,e.dept);

        printf("\nEnter the basic, da and hra:\n");
        scanf("%d%d%d",&e.s.basic,&e.s.da,&e.s.hra);

        gross = e.s.basic + e.s.da + e.s.hra;

        printf("\nEmployee Details.....\n");
        printf("Name: %s\nID: %d\nDept: %s\nGross: %d",e.name,e.id,e.dept,gross);
        return 0;
}
```

Example2: Define a structure to represent Time. Define another structure Flight with following members: name, member of struct Time type. Using this nested structure develop a C Program to read the details of two flights and check whether they are arriving at the same time.

```
int main()
{
    struct Time
    {
        int hour,min,sec;
    };
    struct Flight
    {
        char name[20];
        struct Time t;
    };
    struct Flight f1,f2;
    printf("\nEnter the name and arrival time of first flight:\n");
    scanf("%s%d%d%d",f1.name,&f1.t.hour,&f1.t.min,&f1.t.sec);

    printf("\nEnter the name and arrival time of second flight:\n");
    scanf("%s%d%d%d",f2.name,&f2.t.hour,&f2.t.min,&f2.t.sec);

    if((f1.t.hour==f2.t.hour) && (f1.t.min==f2.t.min) && (f1.t.sec==f2.t.sec))
    printf("\nBoth flights %s and %s are arriving at same time, %d:%d:%d",f1.name,
                                        f2.name, f1.t.hour, f1.t.min, f1.t.sec);
```

```
    else
    {
        printf("\nBoth flights are arriving at different time");
      printf("\nArriving time of first flight %s is %d:%d:%d",f1.name,f1.t.hour,f1.t.min,f1.t.sec);
     printf("\nArriving time of second flight %s is %d:%d:%d",f2.name,f2.t.hour,f2.t.min,f2.t.sec);
    }
    return 0;
}
```

Example3: Define a structure to represent a Point. Define another structure Line with two members of struct Point type. Using this nested structure develop a C Program to read a line and print whether it is a horizontal line, vertical line or oblique line.

```
int main()
{
      struct Point
      {
            int x,y;
       };
       struct Line
       {
            struct Point p1,p2;
       };
       struct Line L;

       printf("\nEnter x & y coordinates of first endpoint of the Line:\n");
       scanf("%d%d",&L.p1.x,&L.p1.y);

       printf("\nEnter x & y coordinates of second endpoint of the Line:\n");
       scanf("%d%d",&L.p2.x,&L.p2.y);

       if(L.p1.y == L.p2.y)
            printf("\nLine is Horizontal");
       else if(L.p1.x == L.p2.x)
            printf("\nLine is vertical");
       else
            printf("\nLine is Oblique");
       return 0;
}
```

Array of Structures

- It is used to maintain the details on *n* records.

Syntax for declaration:

```
struct tagname var[size];
```

Memory allocated for the array, var = sizeof(struct tagname) * size specified

Example:

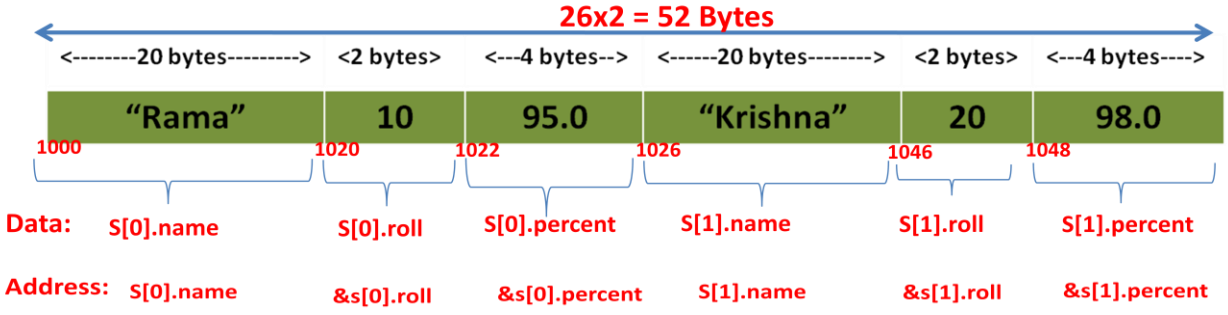
```
struct Student
{
        char name[20];
        int roll;
        float percent;
};
struct Student s[10];
```

Memory allocated for s = sizeof(struct Student) * 10
= 26 bytes * 10 = 260 Bytes

Memory map for Array of Structures

Example: Consider the following structure

```
struct Student
{
        char name[20];
        int roll;
        float percent;
}s[2]={{"Rama",10,95.0},{"Krishna",20,98.0}};
```



To find the base address of an arbitrary record:

Address = Base address of the structure + index * sizeof(struct tagname)

Example:

To find Address of s[1] (i.e., to find &s[1]):

Address = 1000 + 1 * sizeof(struct Student)

= 1000 + 26 = 1026

Example1: C program to print the base address of each record

```
int main()
{
      struct Student
      {
            char name[20];
            int roll;
            float percent;
      }s[2]={{"Rama",10,95.0},{"Krishna",20,98.0}};
      int i;

      printf("\nBase address of each record without using loop:\n");
      printf("%u %u\n",s[0].name,&s[0]);
      printf("%u %u\n",s[1].name,&s[1]));
      printf("\nBase address of each record using loop:\n");
      for(i=0;i<2;i++)
            printf("%u %u\n",s[i].name,&s[i]);
      return 0;
}
```

Example2: Run-time initialization of Array of Structures

```
int main()
{
      struct Student
      {
            char name[20];
            int roll;
            float percent;
      };

      struct Student s[10];
      int i,n;
```

```
    printf("\nEnter the number of students: ");
    scanf("%d",&n);
    printf("\nEnter the details of %d students:\n",n);
    for(i=0;i<n;i++)
    {
        printf("\nEnter name, roll no. and percentage of Student %d:\n",i+1);
        scanf("%s%d%f",s[i].name,&s[i].roll,&s[i].percent);
    }

    printf("\nStudents Details.....\n");
    printf("NAME\tROLL NO.\tPERCENTAGE\n");
    for(i=0;i<n;i++)
        printf("%s\t%d\t%f\n",s[i].name,s[i].roll,s[i].percent);
     return 0;
}
```

Sample program on Pointer to Array of Structure

```
int main()
{
     struct Student
    {
        char name[20];
        int roll;
        float percent;
    };

    struct Student s[10],*p=s;
    int i,n;

    printf("\nEnter the number of students: ");
    scanf("%d",&n);

    printf("\nEnter the details of %d students:\n",n);

    for(i=0;i<n;i++)
    {
        printf("\nEnter name, roll no. and percentage of Student %d:\n",i+1);
        scanf("%s%d%f",p[i].name,&p[i].roll,&p[i].percent);
    }
```

```
    printf("\nStudents Details.....\n");
    printf("NAME\tROLL NO.\tPERCENTAGE\n");
    for(i=0;i<n;i++)
        printf("%s\t%d\t%f\n",p[i].name,p[i].roll,p[i].percent);

    return 0;
}
```

Sample C Programs on Array of structure:

Example1: Display the details of those students whose percentage is in the range 90-100.

```
int main()
{
    struct Student
    {
        char name[20];
        char USN[20];
        char branch[20];
        float percent;
    };
    struct Student s[10];
    int i,n;
    printf("\nEnter the number of students: ");
    scanf("%d",&n);

    printf("\nEnter the details of %d students:\n",n);
    for(i=0;i<n;i++)
    {
        printf("\nEnter name, USN, branch and percentage of Student %d:\n",i+1);
        scanf("%s%s%s%f",s[i].name,s[i].USN,s[i].branch,&s[i].percent);
    }
    printf("\nList of Students whose percentage is in range 90-100.....\n");
    printf("NAME\tUSN\tBRANCH\tPERCENTAGE\n");
    for(i=0;i<n;i++)
    {
        if(s[i].percent >= 90 && s[i].percent <= 100)
        printf("%s\t%s\t%s\t%f\n",s[i].name,s[i].USN,s[i].branch,s[i].percent);
    }
    return 0;
}
```

Example2: Display the details of those students belonging to a specific branch.

```
#include<string.h>
int main()
{
      struct Student
     {
            char name[20];
            char USN[20];
            char branch[20];
            float percent;
     };

     struct Student s[10];
     int i,n;
     char branch[20];

     printf("\nEnter the number of students: ");
     scanf("%d",&n);

     printf("\nEnter the details of %d students:\n",n);
     for(i=0;i<n;i++)
     {
           printf("\nEnter name, USN, branch and percentage of Student %d:\n",i+1);
            scanf("%s%s%s%f",s[i].name,s[i].USN,s[i].branch,&s[i].percent);
     }

     printf("\nEnter the branch to search: ");
     scanf("%s",branch);

     printf("\nList of Students belonging to %s branch ……\n", branch);
     printf("NAME\tUSN\tPERCENTAGE\n");
     for(i=0;i<n;i++)
     {
         if(strcmp(s[i].branch,branch)==0)
               printf("%s\t%s\t%f\n",s[i].name,s[i].USN,s[i].percent);
    }
    return 0;
}
```

Example3: Display the details of those students whose name starting with 'A'.

```
int main()
{
      struct Student
     {
            char name[20];
            char USN[20];
            char branch[20];
            float percent;
     };
     struct Student s[10];
     int i,n;

     printf("\nEnter the number of students: ");
     scanf("%d",&n);

     printf("\nEnter the details of %d students:\n",n);
     for(i=0;i<n;i++)
     {
           printf("\nEnter name, USN, branch and percentage of Student %d:\n",i+1);
            scanf("%s%s%s%f",s[i].name,s[i].USN,s[i].branch,&s[i].percent);
     }

    printf("\nList of Students whose name starting with 'A' ……\n");
    printf("NAME\tUSN\tBRANCH\tPERCENTAGE\n");
    for(i=0;i<n;i++)
   {
        if(s[i].name[0] == 'A')
          printf("%s\t%s\t%s\t%f\n",s[i].name,s[i].USN,s[i].branch,s[i].percent);
   }
   return 0;
}
```

Example4: Display the details of those students belonging to "CSE" and whose percentage >=90.

```
#include<string.h>
int main()
{
      struct Student
     {
            char name[20];
            char USN[20];
            char branch[20];
            float percent;
     };

     struct Student s[10];
     int i,n;

     printf("\nEnter the number of students: ");
     scanf("%d",&n);

     printf("\nEnter the details of %d students:\n",n);
     for(i=0;i<n;i++)
     {
           printf("\nEnter name, USN, branch and percentage of Student %d:\n",i+1);
            scanf("%s%s%s%f",s[i].name,s[i].USN,s[i].branch,&s[i].percent);
     }

    printf("\nList of Students belonging to CSE and percentage>=90 ……\n");
    printf("NAME\tUSN\tPERCENTAGE\n");
    for(i=0;i<n;i++)
   {
        if(strcmp(s[i].branch,"CSE")==0 && s[i].percent >=90)
             printf("%s\t%s\t%f\n",s[i].name,s[i].USN,s[i].percent);
   }

   return 0;
}
```

Example5: Display the details of the student with specified USN.

```
#include<string.h>
int main()
{
      struct Student
     {
            char name[20];
            char USN[20];
            char branch[20];
            float percent;
     };

     struct Student s[10];
     int i,n,flag=0;
     char USN[20];

     printf("\nEnter the number of students: ");
     scanf("%d",&n);

     printf("\nEnter the details of %d students:\n",n);
     for(i=0;i<n;i++)
     {
           printf("\nEnter name, USN, branch and percentage of Student %d:\n",i+1);
           scanf("%s%s%s%f",s[i].name,s[i].USN,s[i].branch,&s[i].percent);
     }

     printf("\nEnter the USN to search: ");
     scanf("%s",USN);

     for(i=0;i<n;i++)
    {
        if(strcmp(s[i].USN,USN)==0)
       {
             printf("\nSearch is successful!!!");
             printf("\nDetails of the student with USN %s:\n",USN);
   printf("NAME: %s\nBRANCH: %s\nPERCENTAGE:%f",s[i].name,s[i].branch,s[i].percent);
           flag=1;
           break;
       }
     }
```

```
    if(flag==0)
        printf("\nFailure, Student with USN %s is not found",USN);
     return 0;
}
```

Example6: Sort the records in ascending order based on the percentage.

```
int main()
{
     struct Student
    {
        char name[20];
        char USN[20];
        char branch[20];
        float percent;
    };
    struct Student s[10],temp;
    int i,j,n;

    printf("\nEnter the number of students: ");
    scanf("%d",&n);
    printf("\nEnter the details of %d students:\n",n);
    for(i=0;i<n;i++)
    {
        printf("\nEnter name, USN, branch and percentage of Student %d:\n",i+1);
         scanf("%s%s%s%f",s[i].name,s[i].USN,s[i].branch,&s[i].percent);
    }

   for(i=0;i<n-1;i++)
   {
        for(j=0;j<n-1-i;j++)
       {
            if(s[j].percent > s[j+1].percent)
            {
                temp = s[j];
                 s[j]=s[j+1];
                 s[j+1]=temp;
            }
       }
   }
```

```
    printf("\nDetails of the students after sorting……\n");
    printf("NAME\tUSN\tBRANCH\tPERCENTAGE\n");
    for(i=0;i<n;i++)
        printf("%s\t%s\t%s\t%f\n",s[i].name,s[i].USN,s[i].branch,s[i].percent);
    return 0;
}
```

Example7: Sort the records in ascending order based on the name of the student.

```
#include<string.h>
int main()
{
     struct Student
    {
        char name[20];
        char USN[20];
        char branch[20];
        float percent;
    };
    struct Student s[10],temp;
    int i,j,n;

    printf("\nEnter the number of students: ");
    scanf("%d",&n);
    printf("\nEnter the details of %d students:\n",n);
    for(i=0;i<n;i++)
    {
        printf("\nEnter name, USN, branch and percentage of Student %d:\n",i+1);
        scanf("%s%s%s%f",s[i].name,s[i].USN,s[i].branch,&s[i].percent);
    }

    for(i=0;i<n-1;i++)
    {
        for(j=0;j<n-1-i;j++)
        {
            if(strcmp(s[j].name,s[j+1].name)>0)
            {
                temp = s[j];
                s[j]=s[j+1];
                s[j+1]=temp;
```

```
                }
            }
        }
        printf("\nDetails of the students after sorting.....\n");
        printf("NAME\tUSN\tBRANCH\tPERCENTAGE\n");
        for(i=0;i<n;i++)
            printf("%s\t%s\t%s\t%f\n",s[i].name,s[i].USN,s[i].branch,s[i].percent);
        return 0;
}
```

Array of Nested Structure

Example: C Program to read the details of *n* employees and to print them.

```
int main()
{
        struct Salary
        {
                int basic;
                int da;
                int hra;
        };
        struct Employee
        {
                char name[20];
                int id;
                char dept[20];
                struct Salary s;
        };
        struct Employee e[10];
        int i,n;

        printf("\nEnter the number of employees: ");
        scanf("%d",&n);

        printf("\nEnter the details of %d employees....\n",n);
        for(i=0;i<n;i++)
        {
                printf("\nEnter the name,id,dept,basic,da,hra of employee %d:\n",i+1);
scanf("%s%d%s%d%d%d",e[i].name,&e[i].id,e[i].dept,&e[i].s.basic,&e[i].s.da,&e[i].s.hra);
        }
```

```
    printf("\nDetails of the Employees.....\n");
    printf("NAME\tID\tDEPT\tBASIC\tDA\tHRA\n");
    for(i=0;i<n;i++)
          printf("%s\t%d\t%s\t%d\t%d\t%d\n",e[i].name,e[i].id,e[i].dept,
                                                e[i].s.basic,e[i].s.da,e[i].s.hra);
    return 0;
}
```

Structures and Functions

Different ways of passing structure to a function:

- Passing individual members of a structure to the function
- Passing entire structure to the function
  - Pass by value (Pass structure variable)
  - Pass by reference (Pass address of structure variable)

Passing individual members of a structure to the function:

- **Function Declaration Syntax:**

```
return_type function_name(datatype1,datatype2,...);
```

- **Function Call Syntax:**

```
var = function_name(member1,member2,...);
```

- **Function Definition Syntax:**

```
return_type function_name(datatype var1,datatype var2,...)
{
        .
        .
        .
    return(var);
}
```

Example: C Program to update the x and y co-ordinate values of a point by 10 using function.

```
struct Point
{
        int x;
        int y;
};

int main()
{
        void update(int,int);      //function declaration
        struct Point p={10,20};
        printf("\nBefore update(), Point is (%d,%d)",p.x,p.y);
        update(p.x,p.y);          //function call
        printf("\nAfter update(), Point is (%d,%d)",p.x,p.y);
        return 0;
}

void update(int a,int b)          //function definition
{
         a = a + 10;
         b = b + 10;
         printf("\nIn update(), Point is (%d,%d)",a,b);
}
```

Note: It is difficult to pass individual members of the structure to the function.

Passing entire structure to the function – Pass by value:

- **Function Declaration Syntax:**

  return_type function_name(struct tagname,...);

- **Function Call Syntax:**

  var = function_name(structure_variable1,...);

- **Function Definition Syntax:**

```
return_type function_name(struct tagname structure_variable1,...)
{
        .
        .
        .
    return(var);
}
```

Example: C Program to update the x and y co-ordinate values of a point by 10 using function.

```
struct Point
{
    int x;
    int y;
};

int main()
{
    struct Point update(struct Point);      //function declaration
    struct Point p={10,20};
    printf("\nBefore update(), Point is (%d,%d)",p.x,p.y);
    p = update(p);                          //function call
    printf("\nAfter update(), Point is (%d,%d)",p.x,p.y);
    return 0;
}

struct Point update(struct Point p)         //function definition
{
    p.x = p.x + 10;
    p.y = p.y + 10;
    printf("\nIn update(), Point is (%d,%d)",p.x,p.y);
    return(p);
}
```

Note:

- We can pass entire structure to the function by passing structure variable as actual argument.
- In the function definition, the formal argument must be of struct tagname type.
- Changes made to the formal argument don't affect the value of the actual argument.
- If changes made to the formal argument need to be affected then return the updated structure variable to the calling function.

Passing entire structure to the function – Pass by reference:

- **Function Declaration Syntax:**

```
return_type function_name(struct tagname *,...);
```

- **Function Call Syntax:**

```
var = function_name(&structure_variable1,...);
```

- **Function Definition Syntax:**

```
return_type function_name(struct tagname *structure_pointer1,...)
{
        .
        .
        .
    return(var);
 }
```

Example: C Program to update the x and y co-ordinate values of a point by 10 using function.

```
struct Point
{
    int x;
     int y;
};
```

```
int main()
{
        void update(struct Point *);      //function declaration
        struct Point p={10,20};

        printf("\nBefore update(), Point is (%d,%d)",p.x,p.y);

        update(&p);                       //function call

        printf("\nAfter update(), Point is (%d,%d)",p.x,p.y);
        return 0;
}

void update(struct Point *p)              //function definition
{
         p->x = p->x + 10;
         p->y = p->y + 10;
         printf("\nIn update(), Point is (%d,%d)",p->x,p->y);
}
```

Note:

- We can pass entire structure to the function by passing address of structure variable as actual argument.
- In the function definition, the formal argument must be a pointer of struct tagname type.
- Changes made to the formal argument will affect the value of the actual argument.

Example: Develop a C Program to read two fractions and compute their product using

a) **Pass by value method**
b) **Pass by reference method**

a) Pass by value method

```
struct fraction
{
        int num;
        int den;
};
```

```
int main()
{
        struct fraction product(struct fraction,struct fraction);
        struct fraction f1,f2,f3;

        printf("\nEnter the numerator and denominator of first fraction:\n");
        scanf("%d%d",&f1.num,&f1.den);

       printf("\nEnter the numerator and denominator of second  fraction:\n");
        scanf("%d%d",&f2.num,&f2.den);

        f3 = product(f1,f2);

        printf("\nf1=%d/%d",f1.num,f1.den);
        printf("\nf2=%d/%d",f2.num,f2.den);
        printf("\nf3=%d/%d",f3.num,f3.den);

        return 0;
}

struct fraction product(struct fraction f1,struct fraction f2)
{
        struct fraction f3;
        f3.num = f1.num * f2.num;
        f3.den = f1.den * f2.den;
        return(f3);
}
```

b) Pass by reference method

```
 struct fraction
 {
          int num;
          int den;
 };

int main()
{
        void product(struct fraction *,struct fraction*,struct fraction *);
        struct fraction f1,f2,f3;
```

```
        printf("\nEnter the numerator and denominator of first fraction:\n");
        scanf("%d%d",&f1.num,&f1.den);

        printf("\nEnter the numerator and denominator of second  fraction:\n");
        scanf("%d%d",&f2.num,&f2.den);

        product(&f1,&f2,&f3);

        printf("\nf1=%d/%d",f1.num,f1.den);
        printf("\nf2=%d/%d",f2.num,f2.den);
        printf("\nf3=%d/%d",f3.num,f3.den);
        return 0;
}

void product(struct fraction *f1,struct fraction *f2,struct fraction *f3)
{
        f3->num = f1->num * f2->num;
        f3->den = f1->den * f2->den;
}
```

Using Both Pass by value and Pass by reference Methods:

```
struct fraction
{
          int num;
          int den;
 };

int main()
{
        void product(struct fraction,struct fraction,struct fraction *);
        struct fraction f1,f2,f3;

        printf("\nEnter the numerator and denominator of first fraction:\n");
        scanf("%d%d",&f1.num,&f1.den);

        printf("\nEnter the numerator and denominator of second  fraction:\n");
        scanf("%d%d",&f2.num,&f2.den);

        product(f1,f2,&f3);
```

```
        printf("\nf1=%d/%d",f1.num,f1.den);
        printf("\nf2=%d/%d",f2.num,f2.den);
        printf("\nf3=%d/%d",f3.num,f3.den);

        return 0;
}

void product(struct fraction f1,struct fraction f2,struct fraction *f3)
{
      f3->num = f1.num * f2.num;
      f3->den = f1.den * f2.den;
 }
```

Example: Develop C functions for the following:

a) **Read a fraction using pass by reference method**
b) **Print a fraction using pass by value method**
c) **Compute product of two fractions. Function must receive two input fractions by value and return resultant fraction**

Using these functions develop a C program to read two fractions and compute their product.

```
 struct fraction
 {
        int num;
        int den;
 };

int main()
{
        void readFraction(struct fraction *);
        void printFraction(struct fraction);
        struct fraction product(struct fraction,struct fraction);
        struct fraction f1,f2,f3;

        printf("\nEnter the numerator and denominator of first fraction:\n");
        readFraction(&f1);
```

```
        printf("\nEnter the numerator and denominator of second  fraction:\n");
        readFraction(&f2);

        f3=product(f1,f2);

        printf("\nFirst fraction, f1=");
        printFraction(f1);

        printf("\nSecond fraction, f2=");
        printFraction(f2);

        printf("\nResultant fraction, f3=");
        printFraction(f3);

        return 0;
}

void readFraction(struct fraction *f)
{
        scanf("%d%d",&f->num,&f->den);
 }

void printFraction(struct fraction f)
{
        printf("%d/%d",f.num,f.den);
}

struct fraction product(struct fraction f1,struct fraction f2)
{
        struct fraction f3;
        f3.num = f1.num * f2.num;
        f3.den = f1.den * f2.den;
        return f3;
}
```

Example: **Define a structure Student with following fields: name, USN, branch and percent. Develop C functions for the following:**
a) Read the details of *n* students
b) Print the details of *n* students
c) Sort *n* records using Bubble sort technique
Using these functions develop a C program to sort the students in descending order based on their percentage of marks scored.

```
 struct Student
{
        char name[20];
        char USN[20];
        char branch[20];
        float percent;
};

int main()
{
        void readRecords(struct Student [],int);
        void printRecords(struct Student [],int);
        void sortRecords(struct Student [],int);
        struct Student s[10];
        int n;

        printf("\nEnter the number of students: ");
        scanf("%d",&n);

        readRecords(s,n);

        printf("\nRecords before sorting.....\n");
        printRecords(s,n);

        sortRecords(s,n);

        printf("\nRecords after sorting.....\n");
        printRecords(s,n);

        return 0;
}
```

```
void readRecords(struct Student s[],int n)
 {
        int i;
        printf("\nEnter the details of %d students:\n",n);
        for(i=0;i<n;i++)
        {
             printf("\nEnter name, USN, branch and percent of student%d:\n",i+1);
            scanf("%s%s%s%f",s[i].name,s[i].USN,s[i].branch,&s[i].percent);
          }
  }

void printRecords(struct Student s[],int n)
{
          int i;
          printf("\nNAME\tUSN\tBRANCH\tPERCENTAGE\n");
          for(i=0;i<n;i++)
               printf("%s\t%s\t%s\t%d\n",s[i].name,s[i].USN,s[i].branch,s[i].percent);

}

void sortRecords(struct Student s[],int n)
{
          int i,j;
          struct Student temp;
          for(i=0;i<n-1;i++)
          {
                for(j=0;j<n-1-i;j++)
                {
                      if(s[j].percent<s[j+1].percent)
                      {
                            temp=s[j];
                            s[j]=s[j+1];
                            s[j+1]=temp;
                       }
                 }
           }
}
```

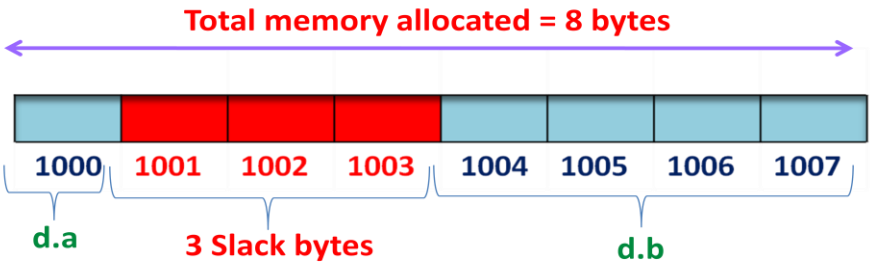
Slack Bytes in Structures

- The Unused extra bytes padded at the end of each member of the structure to maintain word boundary requirements are referred to as **Slack Bytes**.

- **Example:** Consider the following structure:

```
struct Data
{
        char a;
        float b;
}d;
```

Let us assume a 4-byte word boundary for storing the word data in memory.



Example program to print the number of slack bytes in a structure:

```
int main()
{
        struct Data
        {
                char a;
                float b;
         }d;

         printf("\nSize of a = %d Byte",sizeof(d.a));
         printf("\nSize of b = %d Bytes",sizeof(d.b));
         printf("\nSize of d = %d Bytes",sizeof(d));
         printf("\nTotal Slack bytes = %d bytes",sizeof(d) –(sizeof(d.a) + sizeof(d.b)));

         return 0;
}
```

Union

Outline:

- **Definition**
- **Defining a Union**
- **Declaring a Union Variable**
- **Pointer to a Union Variable**
- **Accessing Union Members**
- **Union Initialization**
- **Memory Map for a Union**
- **Array of Union**
- **Nested union**

Union is an user-defined data type in C.

Definition:

It is a collection of data items under a single name in which all data items share the same storage.

Defining a Union

Syntax:

```
union tagname
{
    datatype member1;
    datatype member2;
    . . .
};
```

where, union is a keyword in C which tells the compiler that union is defined
tagname is the name of the union
datatype can be int, float, char etc.
member1, member2,... are called the members/ fields/elements of the union

Example:

```
union Data
{
    char a;
    int b;
    float c;
};
```

Declaring a Union variable

Syntax:

```
union tagname var1,var2,...;
```

Example:

```
union Data d1,d2;
```

Pointer to a Union variable

A variable that holds the address of a union variable is referred to as a pointer to a union variable.

Declaration Syntax:

```
union tagname *ptr_var;
```

Example:

```
union Data d,*p;
```

Initialization Syntax:

```
ptr_var = &union_var;
```

Example:

```
p=&d;
```

Example:

```
union Data
{
        char a;
        int b;
        float c;
};
union Data d,*p=&d;
```

Accessing union members:

- Union members can be accessed only through union variable.

- Three ways to access union members:
  - Using dot operator (for normal union variables)
  - Using indirection operator
  - Using selection operator

  } (for pointer to union variables)

- **Syntax for accessing Union member:**
  - **Using dot operator:** union_variable.union_member
  - **Using indirection operator:** (*ptr_var).union_member
  - **Using selection operator:** ptr_var -> union_member

Example:	dot operator	indirection operator	selection operator
union Data	d.a	(*p).a	p->a
{	d.b	(*p).b	p->b
char a;	d.c	(*p).c	p->c
int b;			
float c;			
}d,*p=&d;			

Union Initialization:

- A union variable can be initialized either during compile time or during run-time.

Compile-time initialization:

- Initializing a union variable during compile time is referred to as compile-time initialization.

Syntax:

```
union tagname
{
        datatype member1;
        datatype member2;
         . . .
};
union tagname var = {val1};
```

Example:

```
union Data
{
        char a;
        int b;
         float c;
};
union Data d = {'A'};
```

Run-time initialization of union variable

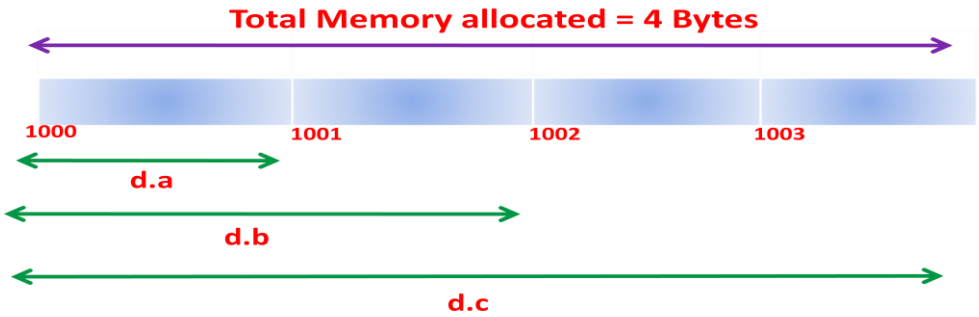
Example program:

```
#include<stdio.h>
int main()
{
	union Data
	{
		char a;
		int b;
		float c;
	};

	union Data d;

	printf("\nEnter a character: ");
	scanf("%c",&d.a);
	printf("\nCharacter entered is %c",d.a);

	printf("\nEnter an integer: ");
	scanf("%d",&d.b);
	printf("\nInteger entered is %d",d.b);

	printf("\nEnter a real number: ");
	scanf("%f",&d.c);
	printf("\nReal number entered is %f",d.c);

	return 0;
}
```

Memory map for Union

Example: Consider the following union definition and declaration:

```
union Data
{
        char a;
         int b;
         float c;
}d;
```



- Here, Address of d.a = Address of d.b = Address of d.c = Address of d = 1000

Array of union

Syntax for declaration:

union tagname var[size];

Memory allocated for the array, var = sizeof(union tagname) * size specified

Example:

```
union Data
{
        char a;
        int b;
        float c;
};
union Data d[10];
```

Memory allocated for d = sizeof(union Data) * 10 = 4 bytes * 10 = **40 Bytes !!!**

Expressions to access Data: d[i].a, d[i].b, d[i].c
Expressions to access Address: &d[i].a, &d[i].b, &d[i].c

Example program to create Generic array using union variable

```
#include<stdio.h>
int main()
{
        union Data
       {
              char a;
               int b;
               float c;
        };
        union Data d[10];
        int i,n,choice,index[10];

        printf("\nEnter the size of the array: ");
        scanf("%d",&n);

       for(i=0;i<n;i++)
       {
            printf("\nEnter your choice: ");
            scanf("%d",&choice);
             switch(choice)
             {
                    case 1: printf("\nEnter a character: ");
                              scanf("%c",&d[i].a);
                              index[i] = choice;
                              break;

                    case 2: printf("\nEnter an integer: ");
                              scanf("%d",&d[i].b);
                              index[i] = choice;
                              break;

                    case 3: printf("\nEnter a real number: ");
                              scanf("%f",&d[i].c);
                              index[i] = choice;
                              break;
                }
          }
```

```
        printf("\nArray Contents:");
        for(i=0;i<n;i++)
        {
              switch(index[i])
              {
                    case 1: printf("\nd[%d] = '%c'",i,d[i].a);
                              break;

                    case 2: printf("\nd[%d] = %d",i,d[i].b);
                              break;

                   case 3: printf("\nd[%d] = %f",i,d[i].c);
                              break;
               }
        }
        return 0;
  }
```

Nested Union

Example1:

```
struct data1
{
      char str[20];
      float x;
      union data2
       {
            int a;
            char b;
        }d2;
}d1;
```

sizeof(d1) = 26 bytes
sizeof(d1.d2) = 2 bytes

Example2:

```
union data1
 {
        char str[20];
        float x;
        union data2
        {
           int a;
           float b;
       }d2;
}d1;
```

sizeof(d1) = 20 bytes
sizeof(d1.d2) = 4 bytes

Example3:

```
union data1
 {
        char str[20];
        float x;
        struct data2
        {
               float a;
               char ch[20];
         }d2;
}d1;
```

sizeof(d1) = 24 bytes
sizeof(d1.d2) = 24 bytes

Example program1: C program to compute square of a number.

```
#include<stdio.h>
#include<stdlib.h>
int main()
{
        struct data1
        {
                int choice;
                union data2
               {
```

```
                int a;
                float b;
            }d2;
        }d1;

         while(1)
        {
            printf("\n\n1:INTEGER\n2:REAL NO.\n3:EXIT");
            printf("\nEnter your choice: ");
            scanf("%d",&d1.choice);

            switch(d1.choice)
           {
                case 1: printf("Enter an integer: ");
                        scanf("%d",&d1.d2.a);
                        printf("Square of %d = %d", d1.d2.a, d1.d2.a* d1.d2.a);
                        break;

                case 2: printf("Enter a real no.: ");
                        scanf("%f",&d1.d2.b);
                        printf("Square of %f = %f", d1.d2.b, d1.d2.b* d1.d2.b);
                        break;

                case 3: exit(0);
               default: printf("Invalid choice");
              }
          }
          return 0;
    }
```

Example program2: C program to compute area of a circle, square and rectangle.

```
#include<stdio.h>
#include<stdlib.h>
#include<math.h>
int main()
{
        union
        {
                struct
                {
```

```
                float radius;
            }circle;
            struct
            {
                float side;
            }square;
            struct
            {
                float length;
                float breadth;
            }rectangle;
        }object;

        int choice;
         while(1)
         {
            printf("\n\n1:Area of Circle\n2:Area of Square\n3:Area of Rectangle\n4:Exit");
            printf("\nEnter your choice: ");
            scanf("%d",&choice);

            switch(choice)
            {
              case 1: printf("Enter the radius: ");
                      scanf("%f",&object.circle.radius);
                      printf("Area = %f",3.142*pow(object.circle.radius,2));
                      break;

               case 2: printf("Enter the side: ");
                       scanf("%f",&object.square.side);
                       printf("Area = %f",pow(object.square.side,2));
                       break;

               case 3: printf("Enter the length and breadth: ");
                       scanf("%f%f",&object.rectangle.length,&object.rectangle.breadth);
                       printf("Area = %f",object.rectangle.length*object.rectangle.breadth);
                      break;

              case 4: exit(0);
              default: printf("Invalid choice");
            }
        }
        return 0;
    }
```

Differences between Structure and Union

Structure	Union
• Keyword struct is used to define structure.	• Keyword union is used to define union.
• Each member within the structure is assigned unique storage area.	• All members share the same storage.
• Each member has different starting address.	• All members have same starting address.
• All members can be accessed simultaneously.	• Only one member can be accessed at a time.
• Modification of any member will not affect the values of other members.	• Modification of member will affect the values of other members.
• Several members can be initialized during compile-time.	• Only first member can be initialized during compile-time.
• Size of structure is greater than or equal to sum of size of each member.	• Size of union is equal to the size of the largest member.
• Less memory efficient.	• More memory efficient particularly in situations where members are not required to be accessed simultaneously.

Bit-Fields

- C supports a special feature called Bit-fields which allows tight packing of information in the memory.

- A Bit-field is a sequence of adjacent bits whose size varies from 1 to 16 bits. (Assume, integer size is 2 bytes)

- **Syntax for defining Bit-fields:**

```
struct tagname
{
        datatype member1:bit_length;
        datatype member2:bit_length;
        ....
};
```

where,

datatype can be either signed or unsigned int only
member1, member2,... are the names of bit-fields
bit_length is the number of bits used to represent the bit-field

- Assume *n* is the bit_length.
  - For unsigned int bit-field, the range of values that can be represented is **0 to 2^n-1**.
  - For signed int bit-field, the range of values that can be represented is **-2^{n-1} to $+2^{n-1}$-1**.

Example1:

```
#include<stdio.h>
int main()
{
    struct Date
   {
       unsigned int date:5;
       unsigned int month:4;
       int year;
  };
  struct Date d = {10,10,2020};
  printf("\nCurrent Date is %d-%d-%d",d.date,d.month,d.year);
  printf("\nSize of d = %d Bytes",sizeof(d));
  return 0;
}
```

Example2:

```
int main()
{
    struct Data
   {
       unsigned int a:3;
       unsigned int b:2;
       int c:2;
       signed int d:3;
       int e:2;
    };
    struct Data x = {13,7,14,7,5};
    printf("\na = %u",x.a);
```

```
        printf("\nb = %u",x.b);
        printf("\nc = %d",x.c);
        printf("\nd = %d",x.d);
        printf("\ne = %d",x.e);
        return 0;
}
```

Output:

```
a = 5
b = 3
c = -2
d = -1
e = 1
```

Example3:

```
#include<stdio.h>
int main()
{
        struct Feedback
        {
                char name[20];
                unsigned int rating:4;
                unsigned int attended:1;
                unsigned int rperson:1;
                unsigned int materials:1;
                unsigned int attendFuture:1;
         };
         struct Feedback f;
         unsigned int x;

         printf("\nEnter your name: ");
         scanf("%s",f.name);

         printf("\nRate the workshop (1-10): ");
         scanf("%u",&x);
         f.rating = x;

         printf("\nHad you attended any such workshop before?1-Yes 0-No: ");
         scanf("%u",&x);
         f.attended = x;
```

```
printf("\nWas the resource person informative?1-Yes 0-No: ");
scanf("%u",&x);
f.rperson = x;

printf("\nWere the materials provided useful?1-Yes 0-No: ");
scanf("%u",&x);
f.materials = x;

printf("\nDo you wish to attend such workshops in future?1-Yes 0-No: ");
scanf("%u",&x);
f.attendFuture = x;

printf("\n**********STUDENT FEEDBACK**********");
printf("\nStudent Name: %s",f.name);
 printf("\nRating: %u",f.rate);

 printf("\nAttended workshop before: ");
 switch(f.attended)
 {
        case 0: printf("No");
                    break;
         case 1:printf("Yes");
                    break;
 }
printf("\nResource person informative: ");
switch(f.rperson)
{
        case 0: printf("No");
                    break;
         case 1:printf("Yes");
                    break;
}
printf("\nMaterials provided useful: ");
switch(f.materials)
{
        case 0: printf("No");
                    break;
         case 1:printf("Yes");
                    break;
}
```

```
    printf("\nWish to attend such workshops in future: ");
    switch(f.attendFuture)
    {
        case 0: printf("No");
                break;
        case 1:printf("Yes");
                break;
    }
    return 0;
}
```

Rules for Bit-fields:

- The bit-field can be of signed or unsigned int type only.
- We cannot get the address of bit-field.
- Bit-fields cannot be arrayed.
- Unnamed bit-fields is allowed.

  Ex: unsigned int:5; // for padding

Command Line Arguments

- These are the arguments passed to the main function by the operating system when the program is invoked.

- To receive command line arguments, the main() function header should be as follows:

```
int main(int argc,char *argv[])          //or **argv
{
      .   .   .
}
```

where,

argc stands for argument count which keeps track of total number of arguments given at the command line.

argv stands for argument vector which is used to point to each of the command line arguments.

- To execute command line arguments program at DOS prompt:

  c:\>CLA_file abc def xyz ↵

- To execute command line arguments program at UNIX terminal:

  $gcc CLA_file.c ↵

  $./a.out abc def xyz ↵

Note: Data read as command line argument will be in string format !!!

Example1: C program to print command line arguments.

```
#include<stdio.h>
int main(int argc, char *argv[])
{
    int i;
    printf("Total number of arguments = %d",argc);
    printf("\nArguments are:\n");
    for(i=0;i<argc;i++)
        printf("argv[%d] = %s\n",i,argv[i]);
    return 0;
}
```

Arguments given at the command line:

C:\>CLA_file abc def ghi ↵

Output:

```
Total number of arguments = 4
Arguments are:
argv[0] = CLA_file
argv[1] = abc
argv[2] = def
argv[3] = ghi
```

Example2: C program to read a string as command line argument and print its length.

```
#include<stdio.h>
#include<string.h>
int main(int argc, char *argv[])
{
        if(argc!=2)
              printf("\nInvalid number of arguments");
        else
              printf("\nLength of the string \"%s\" = %d",argv[1],strlen(argv[1]));
        return 0;
}
```

Arguments given at the command line:

C:\>CLA_file Hello ↵

Output:

Length of the string "Hello" = 5

Example3: C program to read a string as command line argument and check whether it is palindrome or not.

```
#include<stdio.h>
#include<string.h>
int main(int argc, char *argv[])
{
       char str[20];
        if(argc!=2)
              printf("\nInvalid number of arguments");
        else
       {
              strcpy(str,argv[1]);
              strrev(str);
              if(strcmp(argv[1],str)==0)
                    printf("%s is a palindrome",argv[1]);
              else
                    printf("%s is not a palindrome",argv[1]);
        }
        return 0;
}
```

Arguments given at the command line:

C:\>CLA_file BOB ↵

Output:

BOB is a palindrome

Example4: C program to read a string as command line argument and print its first character.

```
#include<stdio.h>
int main(int argc, char *argv[])
{
        if(argc!=2)
                printf("\nInvalid number of arguments");
         else
                printf("\nFirst character of %s is %c",argv[1],argv[1][0]);

          return 0;
}
```

Arguments given at the command line:

C:\>CLA_file Data ↵

Output:

First character of Data is D

Example5: C program to read a string as command line argument and print each character and its corresponding ASCII value.

```
#include<stdio.h>
int main(int argc, char *argv[])
{
        int i;
        printf("\nString is %s\n",argv[1]);
         for(i=0;argv[1][i]!='\0';i++)
                printf("%c - %d\n",argv[1][i],argv[1][i]);

           return 0;
}
```

Arguments given at the command line:

C:\>CLA_file ABC ↵

Output:

String is ABC
A – 65
B – 66
C - 67

Example6: C program to read an integer as command line argument and check whether it is odd or even.

```
#include<stdio.h>
#include<stdlib.h>
int main(int argc, char *argv[])
{
        int num;
        num = atoi(argv[1]);
       if(num%2==0)
              printf("%d is an even number",num);
        else
              printf("%d is an odd number",num);
          return 0;
}
```

Arguments given at the command line:

C:\>CLA_file 25 ↵

Output:

25 is an odd number

Example6: C program to read two integers as command line arguments and compute their sum.

```
#include<stdio.h>
#include<stdlib.h>
int main(int argc, char *argv[])
{
        int num1,num2;
        num1 = atoi(argv[1]);
        num2 = atoi(argv[2]);
```

```
        printf("Sum of %d and %d = %d",num1,num2,num1+num2);
        return 0;
}
```

Arguments given at the command line:

C:\>CLA_file 25   10 ↵

Output:

Sum of 25 and 10 = 35

FILES

Outline:

- **Need for Files**
- **Definition**
- **Declaring a File pointer**
- **Opening and closing a File**
  - **fopen()**
  - **fclose()**
- **I/O operations on files**
  - **fgetc() and fputc()**
  - **getw() and putw()**
  - **fscanf() and fprintf()**
- **Error handling during I/O operations**
  - **ferror()**
  - **feof()**
- **Random access to files**
  - **rewind()**
  - **ftell()**
  - **fseek()**

Need for Files

- It becomes very difficult and time consuming to handle large volume of data through terminals.
- The entire data is lost when the program is terminated or when the computer is turned off.

To overcome these problems, files concept is introduced

Definition:

It is a collection of data bytes stored on disk.

Note: The prototypes for all file functions are defined in the header file, <stdio.h>

Declaration of a file pointer

- To access any file, we need to declare a pointer to FILE structure and then associate it with the particular file. This pointer is referred to as *file* pointer.

- **Syntax for declaring file pointer:**

```
FILE *fp;
```

Here, fp is a pointer to the FILE structure

- **Example:**

```
FILE *fp1,*fp2,*fp3;
```

Opening a file

- fopen() is used to create a new file or open the existing file for use.

- **Syntax :**

```
fp = fopen("filename","mode");
```

or

```
FILE * fopen(const char *,const char *);
```

where,

The first argument indicates the name of the file to be opened for use

The second argument indicates the mode of operation of the file

Mode	operation
"r"	read only
"w"	write only
"a"	append only

On success, fopen() returns a file pointer to the specified file

On failure, it returns NULL

- **Example:**

```
FILE *fp;
fp = fopen("file1.txt","w");
```

Consider the following modes of operations

- **Assume that the file is opened in "w" mode:**
  - If file doesn't exist, fopen() creates a new file with specified name and current pointer is positioned to the beginning of the file.
  - If file already exists then, fopen() opens the file for use while erasing the previous contents and the current pointer is positioned to the beginning of the file.

- **Assume that the file is opened in "r" mode:**
  - If file doesn't exist, fopen() returns NULL.
  - If file already exists then, fopen() opens the file for use with current contents safe and the current pointer is positioned to the beginning of the file.

- **Assume that the file is opened in "a" mode:**
  - If file doesn't exist, fopen() creates a new file with specified name and current pointer is positioned to the beginning of the file.
  - If file already exists then, fopen() opens the file for use with current contents safe and the current pointer is positioned to the end of the file.

Closing a file

- fclose() is used to close the file which was opened for use.

- **Syntax :**

```
int fclose(FILE *);
```

It receives one argument i.e., file pointer to the specified file

On success, fclose() returns zero

On failure, it returns EOF

- When fclose() is executed, all information associated with the file is flushed off from buffer and all the links to the file are broken.

Note: If the file is opened in one mode and we want to operate in other mode then, we have to first close the file and then open the file in the required mode.

Input/Output operations on files

fgetc() and fputc() functions

- **fgetc()** is used to read a single character from the file specified.

- **Syntax :**

```
int fgetc(FILE *);
```

It receives one argument i.e., file pointer to the specified file

On success, it returns the character read from the file

On failure, it returns EOF

- **fputc()** is used to write a single character to the file specified.

- **Syntax :**

```
int fputc(int,FILE *);
```

It receives two arguments. First argument is the character to be written and the second argument is the file pointer to the specified file.

On success, it returns the character written to the file

On failure, it returns EOF

Note: After execution of fgetc() or fputc() function, the current pointer is automatically positioned to the next character in the file.

Program1: ***Read a character and store it in a file. Print the contents of the file.***

Version1:

```
#include<stdio.h>
int main()
{
        FILE * fp;
        char ch;
        printf("\nEnter a character: ");
        scanf("%c",&ch);

        fp = fopen("file1","w");

        fputc(ch,fp);
        printf("\nCharacter \'%c\' is written to the file",ch);
        fclose(fp);
        fp = fopen("file1","r");
        printf("\nCharacter read from the file  is \'%c\'",fgetc(fp));
        fclose(fp);
        return 0;
}
```

Version2:

```
#include<stdio.h>
int main()
{
        FILE * fp;
        char ch;
        printf("\nEnter a character: ");
        scanf("%c",&ch);

        fp = fopen("file1","w+");

        fputc(ch,fp);
        printf("\nCharacter \'%c\' is written to the file",ch);

         rewind(fp);

        printf("\nCharacter read from the file  is \'%c\'",fgetc(fp));
        fclose(fp);
        return 0;
}
```

Program2: ***Read a string and store it in a file. Print the contents of the file.***

```
#include<stdio.h>
int main()
{
        FILE * fp;
        char ch;
        printf("\nEnter a string: \n");
        fp = fopen("file1","w+");

        while((ch=getchar())!='\n')
                fputc(ch,fp);
        printf("\nString is written to the File");

         rewind(fp);
```

```
        printf("\nFile contents:\n");
        while((ch=fgetc(fp))!=EOF)
                printf("%c",ch);
        fclose(fp);
        return 0;
}
```

Integer oriented I/O functions

getw() and putw() functions

- **getw()** is used to read a single integer from the file specified.

- **Syntax :**

  int getw(FILE *);

  It receives one argument i.e., file pointer to the specified file

  On success, it returns the integer read from the file

  On failure, it returns EOF

- **putw()** is used to write a single integer to the file specified.

- **Syntax :**

  int putw(int,FILE *);

  It receives two arguments. First argument is the integer to be written and the second argument is the file pointer to the specified file.

  On success, it returns the integer written to the file

  On failure, it returns EOF

Note: After execution of getw() or putw() function, the current pointer is automatically positioned to the next integer in the file.

Program3: ***Read an integer and store it in a file. Print the contents of the file.***

```
#include<stdio.h>
int main()
{
        FILE * fp;
        int num;
        printf("\nEnter an integer: ");
        scanf("%d",&num);

        fp = fopen("file1","w+");

        putw(num,fp);
        printf("\nInteger %d is written to the file",num);

       rewind(fp);
        printf("\nInteger read from the file  is %d",getw(fp));
        fclose(fp);
        return 0;
}
```

Program4: ***Read n integers and store them in a file. Print the contents of the file.***

```
#include<stdio.h>
#inclue<stdlib.h>
int main()
{
        FILE * fp;
        int i,n,num;
        fp = fopen("file1","w+");
        if(fp==NULL)
        {
                printf("\nError in opening the file");
                exit(0);
        }
       printf("\nEnter the value of n: ");
       scanf("%d",&n);
```

```
        printf("\nEnter %d integers:\n",n);
        for(i=1;i<=n;i++)
        {
                scanf("%d",&num);
                putw(num,fp);
        }

        rewind(fp);

        printf("\nFile contents:\n");
        while((num=getw(fp))!=EOF)
              printf("%d\n",num);

        fclose(fp);
        return 0;
}
```

fscanf() and fprintf() functions

- **fscanf()** is used to read set of data values from the file specified.

- **Syntax :**

  int fscanf(FILE *,const char *,address_list);

  or

  int fscanf(fp,"control string",address_list);

  Here,
  - first argument is the file pointer to the specified file
  - second argument is the control string which indicates the type of value to be read from the file
  - The address of the variables are specified as third, fourth,… arguments

  On success, it returns the number of values successfully read from the file
  On failure, it returns EOF

- **Example:** **fscanf(fp,"%s%d%f",name,&roll,&percent);**

- **fprintf()** is used to write set of data values to the file specified.

- **Syntax :**

int fprintf(FILE *,const char *,variable_list);

or

int fprintf(fp,"control string",variable_list);

Here,

- first argument is the file pointer to the specified file
- second argument is the control string which indicates the type of value to be written to the file
- The variables which hold the values are specified as third, fourth,… arguments

On success, it returns the number of characters written to the file

On failure, it returns EOF

- **Example:** Assume that num=10 and x = 3.5

fprintf(fp,"%d %f",num,x);

Program5: ***Read an integer, real no and a string and store them in a file. Print the contents of the file.***

```
#include<stdio.h>
int main()
{
        FILE * fp;
        int num;
        float x;
        char str[20];
        fp = fopen("file1","w+");
        if(fp==NULL)
        {
                printf("\nError in opening the file");
                exit(0);
        }
```

```
        printf("\nEnter an integer, real no. and a string:\n");
        scanf("%d%f%s",&num,&x,str);
        fprintf(fp,"%d %f %s",num,x,str);

        rewind(fp);
        printf("\nFile contents:\n");
        fscanf(fp,"%d%f%s",&num,&x,str);
        printf("%d\n%f\n%s",num,x,str);
        return 0;
}
```

LAB PROGRAM1: ***Develop a C program to create a sequential file for storing employee records with each record having following information:***

Employee_Id	***Name***	***Department***	***Salary***	***Age***
Non-Zero Positive integer	***25 Characters***	***25 Characters***	***Positive Integer***	***Positive integer***

Write necessary functions to perform the following operations:

a) ***Read the details of a record.***
b) ***Display all the records in the file.***
c) ***Search for specific records based on Department. In case if the required record is not found, suitable message should be displayed.***

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>

struct EMPLOYEE
{
   int empid;
   char name[25];
   char dept[25];
   int salary;
   int age;
}e;
```

```
void add_record(FILE *fp)
{
        printf("\nEnter the details of the employee.....\n");
        printf("ID: ");
        scanf("%d",&e.empid);
        printf("Name: ");
        scanf("%s",e.name);
        printf("Department: ");
        scanf("%s",e.dept);
        printf("Salary: ");
        scanf("%d",&e.salary);
        printf("Age: ");
        scanf("%d",&e.age);
        fprintf(fp,"%d\t%s\t%s\t%d\t%d\n",e.empid,e.name,e.dept,e.salary,e.age);
        printf("\nRecord saved successfully");
}

void display_records(FILE *fp)
{
      printf("ID\t\tNAME\t\tDEPT\t\tSalary\t\tAGE\n");
      printf("-------------------------------------------------\n");

    while((fscanf(fp,"%d%s%s%d%d",&e.empid,e.name,e.dept,&e.salary,&e.age))!=EOF)
       printf("%d\t\t%s\t\t%s\t\t%d\t\t%d\n",e.empid,e.name,e.dept,e.salary,e.age);
}

void search_record(FILE *fp)
{
        int flag=0;
        char dept[20];

        printf("\nEnter the dept to search: ");
        scanf("%s",dept);

    while((fscanf(fp,"%d%s%s%d%d",&e.empid,e.name,e.dept,&e.salary,&e.age))!=EOF)
    {
           if(strcmp(e.dept,dept)==0)
           {
                if(flag==0)
               {
```

```
                    printf("\nSearch Successful !!!");
                    printf("\nID\t\tNAME\t\tDEPT\t\tSalary\t\tAGE\n");
                    printf("-------------------------------------------------------\n");
                    flag=1;
                }

        printf("%d\t\t%s\t\t%s\t\t%d\t\t%d\n",e.empid,e.name,e.dept,e.salary,e.age);
      }
    }
    if(flag==0)
        printf("\nFailure, no such record found !!!");
}

int main()
{
     FILE *fp;
     int choice;
     while(1)
    {
         printf("\n\n1:Add_Record\n2:Search_Record\n3:Display_Records\n4:Exit");
         printf("\nEnter your choice: ");
         scanf("%d",&choice);

         switch(choice)
        {
              case 1: fp=fopen("empfile","a");
                      if(fp==NULL)
                            printf("\nError in opening file");
                      else
                      {
                            add_record(fp);
                            fclose(fp);
                      }
                      break;

               case 2:  fp=fopen("empfile","r");
                        if(fp==NULL)
                              printf("\nError in opening file");
                         else
                        {
```

```
                        search_record(fp);
                        fclose(fp);
                    }
                    break;

            case 3:  fp=fopen("empfile","r");
                     if(fp==NULL)
                          printf("\nNo records to display !!!");
                     else
                     {
                          display_records(fp);
                          fclose(fp);
                     }
                     break;

            case 4:  exit(0);
            default: printf("\nInvalid choice !!!");
        }
    }
    return 0;
}
```

Program6: ***Develop a C program to copy the contents of one file to another.***

```
#include<stdio.h>
int main()
{
        FILE * fp1,*fp2;
        char ch;
        fp1 = fopen("file1","r");
        fp2 = fopen("file2","w+");
        while((ch=fgetc(fp1))!=EOF)
              fputc(ch,fp2);
         printf("\nContents of file1 is copied to file2");

         rewind(fp1);
         rewind(fp2);
```

```
        printf("\nFile1 contents:\n");
        while((ch=fgetc(fp1))!=EOF)
            printf("%c",ch);
        printf("\nFile2 contents:\n");
        while((ch=fgetc(fp2))!=EOF)
            printf("%c",ch);

        fclose(fp1);
        fclose(fp2);
        return 0;
}
```

Program7: ***Develop a C program to copy the contents of one file to another. Read file names as command line arguments.***

```
#include<stdio.h>
int main(int argc,char *argv[])
{
        FILE * fp1,*fp2;
        char ch;
        fp1 = fopen(argv[1],"r");
        fp2 = fopen(argv[2],"w+");

        while((ch=fgetc(fp1))!=EOF)
            fputc(ch,fp2);
        printf("\nContents of file1 is copied to file2");

        rewind(fp1);
        rewind(fp2);

        printf("\nFile1 contents:\n");
        while((ch=fgetc(fp1))!=EOF)
            printf("%c",ch);

        printf("\nFile2 contents:\n");
        while((ch=fgetc(fp2))!=EOF)
            printf("%c",ch);

        fclose(fp1);
```

```
        fclose(fp2);
        return 0;
}
```

Program8: ***Develop a C program to read strings as command line arguments and store them in a file named "File1". Store the strings starting with letter 'A' to another file named "File2". Print the contents of both the files.***

```
#include<stdio.h>
int main(int argc,char *argv[])
{
        FILE * fp1,*fp2;
        int i;
        char str[20];

        if(argc<2)
        {
                printf("\nInvalid number of arguments");
                exit(0);
        }

        fp1 = fopen("File1","w+");
        fp2 = fopen("File2","w+");

        for(i=1;i<argc;i++)
        {
                fprintf(fp1,"%s\n",argv[i]);

                if(argv[i][0] == 'A')
                        fprintf(fp2,"%s\n",argv[i]);
        }
        rewind(fp1);
        rewind(fp2);

        printf("\nFile1 contents:\n");
        while(fscanf(fp1,"%s",str)!=EOF)
                printf("%s\n",str);
```

```
        printf("\nFile2 contents:\n");
        while(fscanf(fp2,"%s",str)!=EOF)
                printf("%s\n",str);

        fclose(fp1);
        fclose(fp2);
        return 0;
}
```

Program9: ***Develop a C program to read integers as command line arguments and store them in a file named "Integers". Store the odd numbers in a file named "Odd" and even numbers in a file named "Even". Print the contents of all the three files.***

```
#include<stdio.h>
#include<stdlib.h>
int main(int argc,char *argv[])
{
        FILE * fp1,*fp2,*fp3;
        int num,i;
        if(argc<2)
        {
                 printf("\nInvalid number of arguments");
                 exit(0);
        }

        fp1 = fopen("Integers","w+");
        fp2 = fopen("Even","w+");
        fp3 = fopen("Odd","w+");

        for(i=1;i<argc;i++)
        {
                num=atoi(argv[i]);
                putw(num,fp1);
                if(num%2==0)
                        putw(num,fp2);
                else
                        putw(num,fp3);
        }
```

```
        rewind(fp1);
        rewind(fp2);
        rewind(fp3);

        printf("\nFile containing input integers:\n");
        while((num=getw(fp1))!=EOF)
                printf("%d\t",num);

        printf("\nFile containing Even integers:\n");
        while((num=getw(fp2))!=EOF)
                printf("%d\t",num);

        printf("\nFile containing Odd integers:\n");
        while((num=getw(fp3))!=EOF)
                printf("%d\t",num);

        fclose(fp1);
        fclose(fp2);
        fclose(fp3);
        return 0;
}
```

Program10: Develop a C program to read strings as command line arguments and store them in a file named "Strings". Store the length of each string in a file named "Lengths". Print the contents of both the files.

```
#include<stdio.h>
#include<string.h>
int main(int argc,char *argv[])
{
        FILE * fp1,*fp2;
        int num,i;
        char str[20];
        if(argc<2)
        {
                printf("\nInvalid number of arguments");
                exit(0);
        }
```

```
        fp1 = fopen("Strings","w+");
        fp2 = fopen("Lengths","w+");

        for(i=1;i<argc;i++)
        {
                fprintf(fp1,"%s\n",argv[i]);
                putw(strlen(argv[i]),fp2);
        }

        rewind(fp1);
        rewind(fp2);

        printf("\nSTRINGS\t\t\tLENGTHS\n");
        printf("-----------------------------------------------------------\n");
        while(fscanf(fp1,"%s",str)!=EOF &&  (num=getw(fp2))!=EOF)
                printf("%s\t\t\t%d\n",str,num);

         fclose(fp1);
         fclose(fp2);
         return 0;
}
```

Error Handling in Files

Typical error situations:

- Trying to use the file which is not opened.
- Opening the file in one mode and trying to operate in other mode.
- Trying to read beyond end of file.
- Trying to write to write protected file (Read-only file).

ferror():

- It is used to check whether any error has occurred during I/O operation on a specified file.

- **Syntax:**

```
int ferror(FILE *);
```

It receives one argument i.e., file pointer to the specified file.
It returns a non-zero value if error has occurred.
Otherwise, it returns zero

Sample program on ferror():

```
#include<stdio.h>
int main()
{
        FILE *fp;
        char ch;

        fp=fopen("file1","w");
        fputc('A',fp);
        printf("Character \'A\' is written to the file");

        rewind(fp);
        ch= fgetc(fp);

        if(ferror(fp))
              printf("\nError in reading from the file !!!");
        else
               printf("\nCh = %c",ch);
        fclose(fp);
        return 0;
}
```

feof():

- It is used to check whether end of file is reached or not during I/O operation on a specified file.

- **Syntax:**

```
int feof(FILE *);
```

It receives one argument i.e., file pointer to the specified file
It returns a non-zero value if end of file is reached
Otherwise, it returns zero

Sample program on feof():

```
#include<stdio.h>
int main()
{
        FILE *fp;
        int i,n,num;

        fp=fopen("file1","w+");
        printf("Enter the value of n: ");
        scanf("%d",&n);
        printf("\nEnter %d integers:\n",n);
        for(i=1;i<=n;i++)
        {
                scanf("%d",&num);
                putw(num,fp);
        }
        rewind(fp);

        printf("\nFile Contents:\n");
        while(1)
        {
                num=getw(fp);
                if(feof(fp))
                      break;
                printf("%d\n",num);
        }

        fclose(fp);
        return 0;
}
```

Random Access to Files

rewind():

- It is used to set the current pointer position to the beginning of the file.

- **Syntax:**

```
void rewind(FILE *);
```

It receives one argument i.e., file pointer to the specified file
It doesn't return any value.

Sample program on rewind():

```
#include<stdio.h>
int main()
{
        FILE *fp;

        fp=fopen("file1","w+");
        fputc('A',fp);
        printf("Character \'A\' is written to the file");

       rewind(fp);
       printf("\nCharacter read from the file is \'%c\'",fgetc(fp));
       fclose(fp);
       return 0;
}
```

ftell():

- It is used to get the current pointer position in the specified file.

- **Syntax:**

  long ftell(FILE *);

  It receives one argument i.e., file pointer to the specified file

  On success, it returns the number of bytes read or written so far in the specified file

  On failure, it returns EOF

Sample program on ftell():

```
#include<stdio.h>
int main()
{
        FILE *fp;

        fp=fopen("file1","w+");
```

```
        fputc('A',fp);
        fputc('B',fp);
        printf("No. of bytes written to the file so far = %ld bytes",ftell(fp));

        rewind(fp);
        printf("\n\nFirst character read from the file is \'%c\'",fgetc(fp));
        printf("\n\nNo. of bytes read from the file so far = %ld byte",ftell(fp));
        fclose(fp);
        return 0;
}
```

fseek():

- It is used to set the current pointer position to the desired point in the specified file.

- **Syntax:**

  int fseek(FILE *,long,int);

  It receives three arguments:

  - **First argument** is the file pointer to the specified file
  - **Second argument** is the offset which indicates the number of bytes by which the current pointer is to be moved
    - It can be positive or negative
      - +ve to move the current pointer in the forward direction
      - -ve to move the current pointer in the reverse direction
  - **Third argument** indicates the position from where the current pointer is to be moved

Macros	Whence(Position)	Meaning
SEEK_SET	**0**	**Beginning of the file**
SEEK_CUR	**1**	**Current position**
SEEK_END	**2**	**End of the file**

On success, it returns zero
On failure, it returns a non-zero value

What is the operation performed in the following C statements?

- **fseek(fp,0L,0);**

  Set the current pointer to the beginning of the file and move it by zero bytes in the forward direction (fseek(fp,0,0); is equivalent to rewind(fp);)

- **fseek(fp,0L,1);**

  Move the current pointer by zero bytes in the forward direction from the current position

- **fseek(fp,0L,2);**

  Set the current pointer to the end of the file and move it by zero bytes in the forward direction

- **fseek(fp,*m*,0);**

  Set the current pointer to the beginning of the file and move it by *m* bytes in the forward direction

- **fseek(fp,*m*,SEEK_CUR)**

  Move the current pointer by *m* bytes in the forward direction from the current position

- **fseek(fp,-*m*,1);**

  Move the current pointer by *m* bytes in the reverse direction from the current position

- **fseek(fp,-*m*,SEEK_END);**

  Set the current pointer to the end of the file and move it by *m* bytes in the reverse direction

- **fseek(fp,5,SEEK_SET);**

  Set the current pointer to the beginning of the file and move it by 5 bytes in the forward direction

- **fseek(fp,-3L,2)**

  Set the current pointer to the end of the file and move it by 3 bytes in the reverse direction

Sample programs on fseek()

Program1: ***To find the number of bytes in the file.***

```
#include<stdio.h>
int main()
{
        FILE *fp;

        fp=fopen("file1","r");

        fseek(fp,0,2);
       printf("\nFile  contains %ld bytes",ftell(fp));
       fclose(fp);
       return 0;
}
```

Program2: ***To print alternate characters in the file.***

```
#include<stdio.h>
int main()
{
        FILE *fp;
        char ch;
        fp=fopen("file1","r");

       printf("Alternate characters in the file:\n");
        while((ch=fgetc(fp))!=EOF)
        {
                printf("%c",ch);
                fseek(fp,1,1);
        }
        fclose(fp);
        return 0;
}
```

Program3: ***To swap first and last character in the file.***

```
#include<stdio.h>
int main()
{
        FILE *fp;
        char ch1,ch2;
        int n;

       fp=fopen("file1","r+");
       ch1 = fgetc(fp);
       fseek(fp,-1,2);
       ch2 = fgetc(fp);
       rewind(fp);
       fputc(ch2,fp);
       fseek(fp,-1,2);
       fputc(ch1,fp);
       rewind(fp);

       printf("File Contents after swapping first and last characters:\n");
       while((ch1=fgetc(fp))!=EOF)
            printf("%c",ch1);

        fclose(fp);
        return 0;
}
```

Program4: ***To print the contents of the file in reverse order.***

Version1:

```
#include<stdio.h>
int main()
{
        FILE *fp;
        int n=1;

        fp=fopen("file1","r");

        printf("File contents in reverse order:\n");
```

```
        while(!fseek(fp,-n,2))
        {
                printf("%c",fgetc(fp));
                n++;
         }
         fclose(fp);
         return 0;
}
```

Version2:

```
#include<stdio.h>
int main()
{
         FILE *fp;
         int n=1;

        fp=fopen("file1","r");
        fseek(fp,0,2);
        printf("File contents in reverse order:\n");
        while(!fseek(fp,-n,1))
        {
                printf("%c",fgetc(fp));
                n=2;
         }
         fclose(fp);
         return 0;
}
```

********** **END OF UNIT - 1** **********