CHARUSAT
CHAROTAR UNIVERSITY OF SCIENCE AND TECHNOLOGY

cspit
Chandubhai S Patel Institute of Technology

CE143: COMPUTER CONCEPTS & PROGRAMMING

UNIT-10
Structure and Union

N. A. Shaikh
nishatshaikh.it@charusat.ac.in

Topics to be covered

- **Need of user-defined data type**
- **Structure definition**
- **Declaration and Initialization of variables**
- **Array as member**
- **Array of structure variables**
- **Structure within structure**
- **Structure as function arguments**
- **Union**

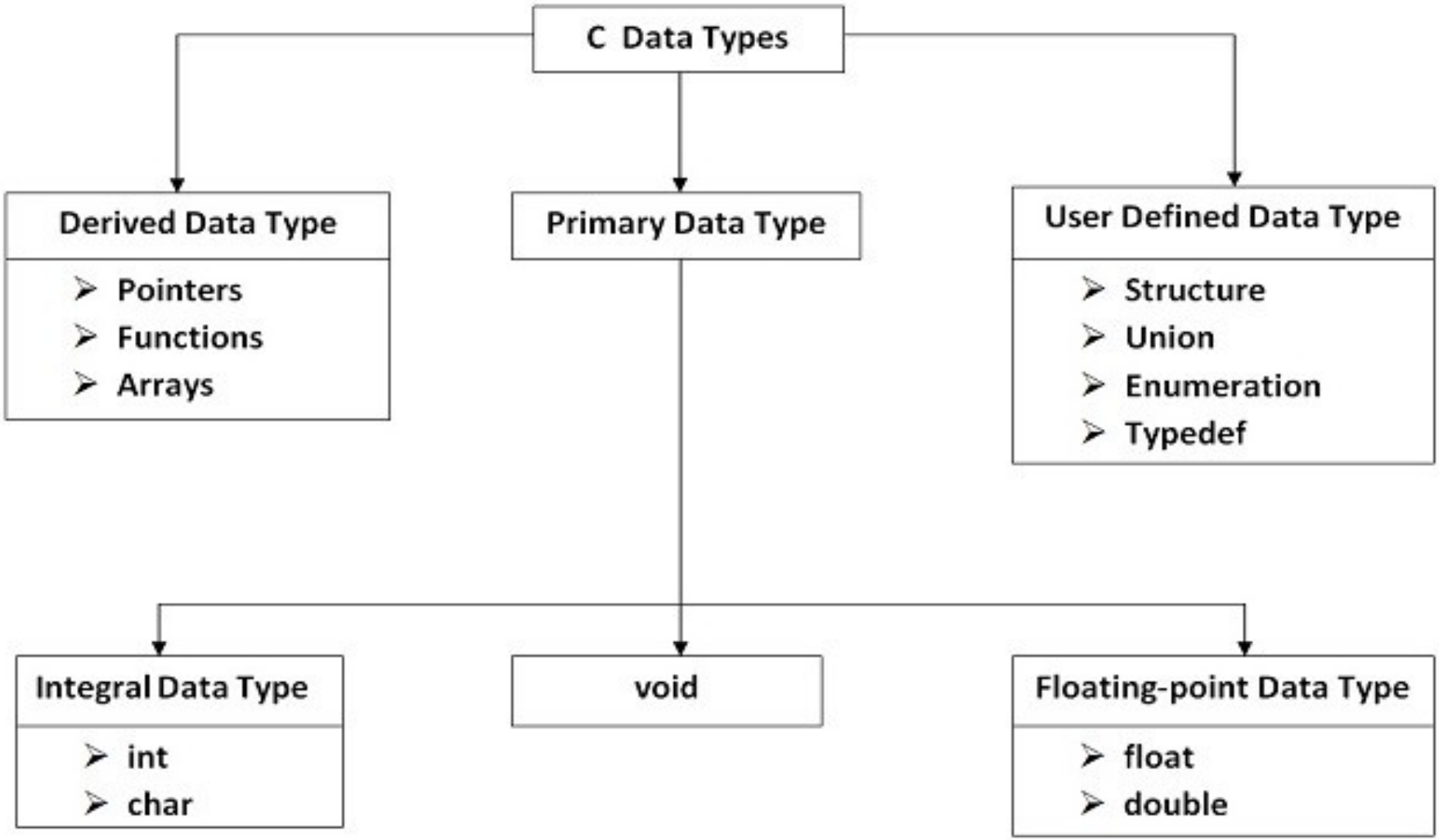
Need of user-defined data type

- Sometimes, the basic set of data types defined in the C language such as **int, float** etc. may be **insufficient** for your application.

- In such circumstances, the programmer can **invent his/her own data types** using user-defined data types

- Data types which are **defined by the user as per his/her will** are called **user-defined data types**.

C Data Types


C Data Types
Derived Data Type
Pointers
Functions
Arrays
Primary Data Type
User Defined Data Type
Structure
Union
Enumeration
Typedef
Integral Data Type
int
char
void
Floating-point Data Type
float
double

- **Arrays** can be used to represent a group of data items that belong to the **same data type**.
- What if we want to represent a collection of data items of different types using single name?
- C supports a constructed data type known as **structures**, a mechanism for packing data of **different types.**

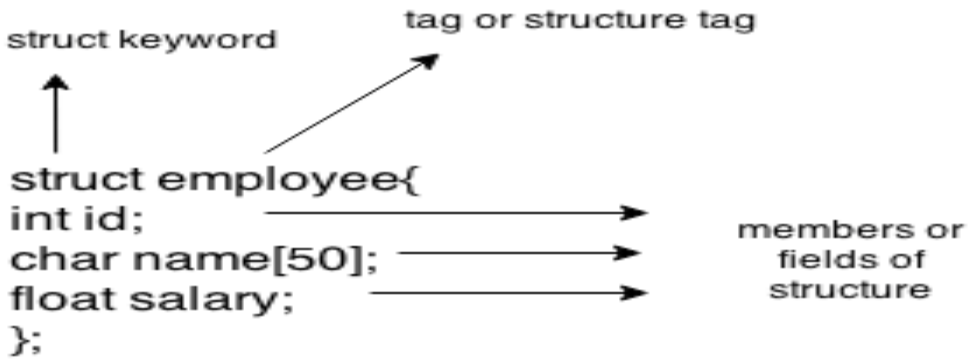
- **structure** is **user defined data type** that allows to combine data of different types together
- Structures are used to represent a **record**.
- Suppose you want to keep track of your **books** in a library. You might want to track the following attributes about each book –
  - Title
  - Author
  - Pages
  - Book ID
  - Price

- Structures must be defined first for their format that may be used later to declare structure variables.

Syntax:

```
struct tag_name
{
        data_type member1;
        data_type member2;
        ..........
        ..........
};
```

Example:



NOTE:

- The template is terminated with a **semicolon**.
- The individual members can be ordinary variables, pointers, arrays, or other structures (any data type)
- The **member names** within a particular structure must be **distinct** from one another
- A member name can be the same as the name of a variable defined outside of the structure
- The tag name such as employee can be used to declare structure variables of its type, later in the program

After defining a structure format we can declare variables of that type.

It includes :

1. The keyword struct
2. The structure tag name
3. List of variable names separated by commas
4. A terminating semicolon

```
struct employee e;
```

OR

```
struct employee e1,e2,e3;
```

Here e,e1,e2 and e3 are variables of type struct employee

NOTE:

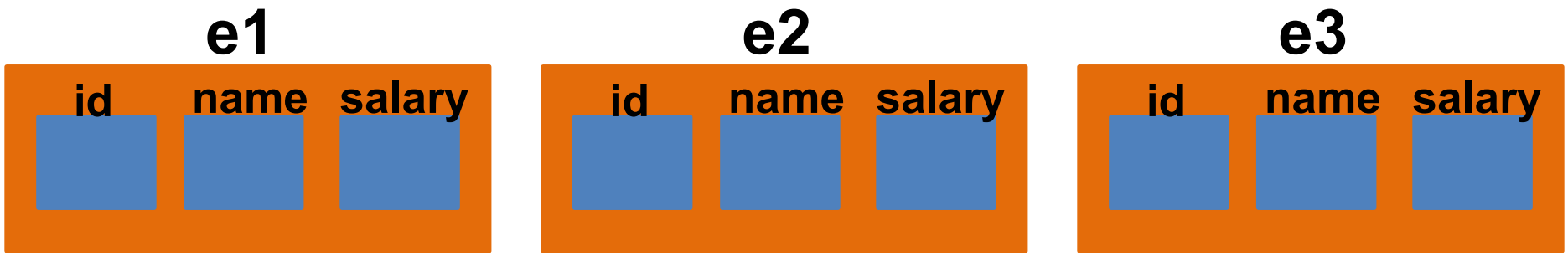
- Each one of these variables have 3 members as specified by the template
- Members of a structure themselves are not variables
- **Members do not occupy any memory until they are associated with the structure variables**
- When compiler comes across a declaration statement, it reserves memory space for the structure variables.

Method 1

```
struct employee
{
    int id;
    char name[50];
    float salary;
}e1,e2,e3;

void main()
{

}
```

Method 2

```
struct employee
{
    int id;
    char name[50];
    float salary;
};

void main()
{
    struct employee e1,e2,e3;
}
```



Local structure & Global structure

```c
//LOCAL: Definition inside function
void main()
{
    struct LOCAL
    {
        int a,b;
    };
    struct LOCAL l;
}

void check()
{
    struct LOCAL l;   //ERROR
}
```

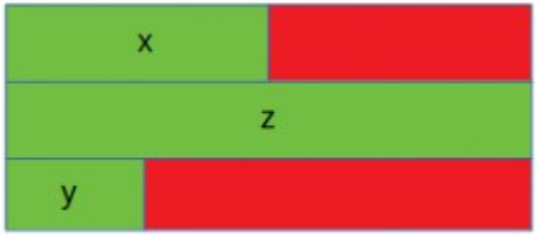
```c
//GLOBAL: Definition outside to all the function
struct GLOBAL
{
        int a,b;
};

void main()
{
    struct GLOBAL g;
}

void check()
{
    struct GLOBAL g;
}
```

- The sizeof for a struct is not always equal to the sum of sizeof of each individual member.
- This is because of the **padding** added by the compiler to avoid alignment issues.
- Padding is only added when a structure member is followed by a member with a **larger size** or at the end of the structure.
- Different compilers might have different alignment constraints as C standards state that alignment of structure totally depends on the implementation.

Size of struct

case 1

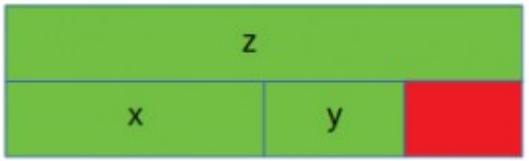
```c
void main()
{
    struct A
    {
        int x;
        double z;
        short int y;
    };
    printf("Size=%d", sizeof(struct A));
}
```




```
Size=24
```

case 2

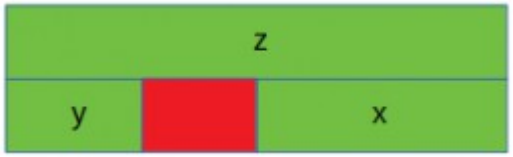
```c
void main()
{
    struct A
    {
        double z;
        int x;
        short int y;
    };
    printf("Size=%d", sizeof(struct A));
}
```




```
Size=16
```

case 3

```
void main()
{
    struct A
    {
        double z;
        short int y;
        int x;
    };
    printf("Size=%d", sizeof(struct A));
}
```




```
Size=16
```

NOTE:

In order to minimize the amount of padding, the struct members must be sorted in a descending order(similar to the case 2).

case 4

```
void main()
{
    struct A
    {
        char z;
        int x;
        short int y;
    };
printf("Size=%d", sizeof(struct A));
}
```

z	1	2	3
x	x	x	x
y	y	1	2

```
Size=12
```

case 5

```
void main()
{
    struct A
    {
        char z;
        short int y;
        int x;
    };
printf("Size=%d", sizeof(struct A));
}
```

x	1	2	3
y	y	1	2
z	z	z	z

```
Size=8
```

- Structure members **cannot be initialized with declaration**.

For example the following C program **fails in compilation**.

```
struct st_record
{
    int weight=60;   //ERROR
    float height=170.75;   //ERROR
};
void main()
{

}
```

NOTE:

- The reason for above error is simple, when a datatype is declared, no memory is allocated for it.
- Memory is allocated only when variables are created.

Method 1

```
void main()
{
    struct
    {
        int weight;
        float height;
    }student={60,180.75};
}
```

Method 2

```
void main()
{
    struct st_record
    {
        int weight;
        float height;
    };
    struct st_record student1={60,180.75};
    struct st_record student2={53,170.60};
}
```

Method 3

```
struct st_record
{
        int weight;
        float height;
}student1={60,180.75};

void main()
{
    struct st_record student2={53,170.60};
}
```

Rules for Initializing Structure:

1. We cannot initialize individual members inside the structure template
2. The order of values enclosed in braces must match the order of members in the structure definition
3. It is permitted to have a partial initialization. We can initialize only the first few members and leave the remaining blank. The uninitialized members should be only at the end of the list.
4. The uninitialized members will be assigned default values as follows:
   - **Zero for integers and floating point numbers**
   - **'\0' for characters and strings**

- Members are not variables
- They should be linked to the structure variables in order to make them meaningful members.
- For example, the word salary has no meaning whereas the phrase '**salary of e1**' has a meaning
- The link between a member and a variable is established using the member operator '.'
- Also known as '**dot operator**' or '**period operator**'

```
e1.salary
```

- Assigning values to the members of e1:

```
e1.id=001;
e1.name="Nishat";
e1.salary=75000.50;
strcpy(e1.name,"Shaikh");
```

- We can also use scanf

```
scanf("%d",&e1.id);
scanf("%s",e1.name);
scanf("%f",&e1.salary);
```

- **Struct keyword** is used to declare structure.
- Members of structure are enclosed within opening and closing braces.
- Declaration of Structure reserves no space.
- It is nothing but the “ Template / Map / Shape ” of the structure .
- Memory is created , very first time when the variable is created /Instance is created.

Size of struct

```c
#include <stdio.h>
struct employee
{
    int id;  //4
    char name[30];  //30
};

void main()
{
    struct employee e;
    printf("Size of employee=%d\n", sizeof(struct employee));
    printf("Size of employee=%d\n", sizeof(e));
    printf("Size of id=%d\n", sizeof(e.id));
    printf("Size of name=%d\n", sizeof(e.name));
}
```

```
Size of employee=36
Size of employee=36
Size of id=4
Size of name=30
```

Defining a structure type, struct personal that would contain person name, date of joining and salary. Using this structure, write a program to read this information for one person from the keyboard and print the same on the screen.

```c
struct personal
{
    char name[20];
    int day;
    char month[10];
    int year;
    float salary;
};

void main()
{
    struct personal person;

    printf("Input Values\n");
    scanf("%s %d %s %d %f",person.name,&person.day,person.month,&person.year,&person.salary);
    printf("%s %d %s %d %f",person.name,person.day,person.month,person.year,person.salary);
}
```

```
Input Values
N.A.Shaikh 2 Feb 1993 75000
N.A.Shaikh 2 Feb 1993 75000.000000
```

Create a Structure called library to hold accession number, title of the book, author name, price of the book and flag indicating whether the book is issued or not. (flag = 1 if the book is issued, flag = 0 otherwise). Write a program to enter data of one book and display the data. Write this same program with Union also.

```c
#include<stdio.h>
struct library
{
    int accession_number;
    char title_of_the_book[10], author_name[10];
    int price_of_the_book;
    int flag;
}s;

void main()
{
    printf("Enter Data:\n");

    printf("Enter accession_number: ");
    scanf("%d", &s.accession_number);
    printf("Enter title_of_the_book: ");
    scanf("%s", s.title_of_the_book);
    printf("Enter author_name: ");
    scanf("%s", s.author_name);
    printf("Enter price_of_the_book: ");
    scanf("%d", &s.price_of_the_book);
    printf("press 1 if book is issued and 0 if available\n");
    scanf("%d",&s.flag);
    printf("\nDisplaying Data:\n");

    printf("accession_number: %d\n",s.accession_number);
    printf("title_of_the_book:%s\n",s.title_of_the_book);
    printf("author_name:%s\n",s.author_name);
    printf("price_of_the_book: %d\n",s.price_of_the_book);
    if(s.flag==0)
        printf("Book is available\n");
    else
        printf("Book is issued\n");
}
```

```
Enter Data:
Enter accession_number: 001
Enter title_of_the_book: CCP
Enter author_name: Nishat
Enter price_of_the_book: 500
press 1 if book is issued and 0 if available
1

Displaying Data:
accession_number: 1
title_of_the_book:CCP
author_name:Nishat
price_of_the_book: 500
Book is issued
```

Let’s Practice

Define a structure DATE_STRUCT containing three members: integer month, integer date and integer year. Develop a program that would assign values to the individual member and display the date in following form: 21/12/2016. Also check that year is leap year or not.

Define a structure called **TimeStruct** having 3 members called Hour, Minute and Second. Develop a program that would assign values to the individual members and display the time in the form **10:40:30.**

Array Vs Structures

Array	Structures
Array refers to a collection consisting of elements of homogenous(same) data type.	Structure refers to a collection consisting of elements of heterogeneous(different) data type.
Array uses subscripts or “[ ]” (square bracket) for element access	Structure uses “.” (Dot operator) for element access
Array is pointer as it points to the first element of the collection.	Structure is not a pointer
Array size is fixed and is basically the number of elements multiplied by the size of an element.	Structure size is not fixed as each element of Structure can be of different type and size.
Bit filed is not possible in an Array.	Bit filed is possible in an Structure.
Array declaration is done simply using [] and not any keyword.	Structure declaration is done with the help of “struct” keyword.
Arrays is a derived datatype	Structure is a user-defined datatype.

Array Vs Structures (Cont..)

Array	Structures
Array traversal and searching is easy and fast.	Structure traversal and searching is complex and slow.

data_type array_name[size];	struct sruct_name{
	data_type1 ele1;
	data_type2 ele2;
	};

Array elements are stored in continuous memory locations.	Structure elements may or may not be stored in a continuous memory location.
Array elements are accessed by their index number using subscripts.	Structure elements are accessed by their names using dot operator.
Any array behaves like a built-in data types. All we have to do is to declare an array variable and use it.	In the case of structure, first, we have to design and declare a data structure before the variable of that type are declared and used.
Array allocates static memory.	Structures allocate dynamic memory.

Copying and Comparing Structure Variables

- Two variables of the same structure type can be copied the same way as ordinary variables.
- If person1 and person2 belong to the same structure,

```
person1=person2;
person2=person1;
```

valid

- However C does not permit any logical operations on structure variables

```
person1==person2
person1!=person2;
```

Invalid

- We can compare them by comparing members **individually**

Copying and Comparing Structure Variables

```
struct class
{
    int number;
    char name[20];
    float marks;
};

void main()
{
    int x;
    struct class student1={111,"Rao",72.50};
    struct class student2={222,"Reddy",67.00};
    struct class student3;

    student3=student2;

    x=((student3.number==student2.number) && (student3.marks==student2.marks))?1:0;

    if(x==1)
    {
        printf("\nstudent2 and student3 are same\n\n");
        printf("%d %s %f\n",student3.number,student3.name,student3.marks);
    }
    else
        printf("\nstudent2 and student3 are different\n\n");
}
```

```
student2 and student3 are same

222 Reddy 67.000000
```

Copying and Comparing Structure Variables

Define a structure Distance having data members: km and meter. Write a program that declares two structure variables, enter data and check whether the two distances are same or not.

Members can be manipulated using expressions and operators

```
if(student1.number==111)
    student1.marks+=10.00;

float sum=student1.marks+student2.marks;

student2.marks*=0.5;

student1.number++;
++student1.number;
```

- The precedence of the member operator is higher than all arithmetic and relational operators and therefore no parenthesis are required.

A structure has data members: cm and mm. Write a program that enter two variables, add them and assign into third variable. Validate and print the answer.

(e.g. 16mm = 1 cm and 6 mm)

```c
#include <stdio.h>

struct Distance
{
    int cm;
    float mm;
} d1, d2, result;

void main()
{
    // take first distance input
    printf("Enter 1st distance\n");
    printf("Enter cm: ");
    scanf("%d", &d1.cm);
    printf("Enter mm: ");
    scanf("%f", &d1.mm);

    // take second distance input
    printf("\nEnter 2nd distance\n");
    printf("Enter cm: ");
    scanf("%d", &d2.cm);
    printf("Enter mm: ");
    scanf("%f", &d2.mm);

    // adding distances
    result.cm = d1.cm + d2.cm;
    result.mm = d1.mm + d2.mm;

    // convert mm to cm if greater than 10
    while (result.mm >= 10.0) {
        result.mm = result.mm - 10.0;
        ++result.cm;
    }
    printf("\nSum of distances =%d cm & %.1f mm", result.cm, result.mm);
}
```

```
Enter 1st distance
Enter cm: 2
Enter mm: 4

Enter 2nd distance
Enter cm: 3
Enter mm: 7

Sum of distances = 6 cm & 1.0 mm
```

Let’s Practice

Define a structure Time having integer data members hour, minute, second. Write a program to enter and then add two variables and store the result into third variable. Validate the seconds and minutes of the result and print it.
(e.g.3 Hour 65 Min 70 Sec= 4 Hour 6 Min 10 Sec)

A structure has data members: meter and cm. Write a program that enter two variables, add them and assign into third variable. Validate and print the answer.
(e.g. 120cm=1 meter and 20 cm)

We can use structure and array as :

- Array of Structure
- Array within Structure(Array as member)

- Consider a case, where we need to store the data of 100 students.
- Declaring 100 different structure variables and store them one by one will be tough.
- However, c enables us to declare an **array of structures**
- Declaring an array of structure is same as declaring an array of fundamental types.
- Since an array is a collection of elements of the same type. In an array of structures, each element of an array is of the structure type.
- Also known as **collection of structure** or **structure array**

- Here, each element of array represents a structure variable

Example:

```
struct Class student[100];
```

Here struct Class is data type; student is array name and elements are 100

```
struct marks
{
    int subject1;
    int subject2;
    int subject3;
};
void main()
{
    struct marks student[3]={{45,68,81},{75,53,69},{57,36,71}};
}
```

.subject2	68
.subject3	81
student[1].subject1	75
.subject2	53
.subject3	69
student[2].subject1	57
.subject2	36
.subject3	71

Example: Array of Structures

```c
#include<stdio.h>
struct student
{
    int rollno;
    char name[10];
};
void main()
{
    int i;
    struct student st[5];

    printf("Enter Records of 5 students");
    for(i=0;i<5;i++)
    {
        printf("\nEnter Rollno:");
        scanf("%d",&st[i].rollno);
        printf("Enter Name:");
        scanf("%s",&st[i].name);
    }

    printf("\nStudent Information List:");
    for(i=0;i<5;i++)
    {
        printf("\nRollno:%d, Name:%s",st[i].rollno,st[i].name);
    }
}
```

```
Enter Records of 5 students
Enter Rollno:1
Enter Name:Mufid

Enter Rollno:2
Enter Name:Rency

Enter Rollno:3
Enter Name:Honey

Enter Rollno:4
Enter Name:Heli

Enter Rollno:5
Enter Name:Drashti

Student Information List:
Rollno:1, Name:Mufid
Rollno:2, Name:Rency
Rollno:3, Name:Honey
Rollno:4, Name:Heli
Rollno:5, Name:Drashti
```

- A structure may contain elements of different data types int, char, float, double, etc.
- It may also contain an **array as its member**.
- Such an array is called an **array within a structure**.
- An array within a structure is a member of the structure and can be accessed just as we access other elements of the structure.

Array within Structures

```c
struct Student
{
	int Roll;
	char Name[25];
	int Marks[3];
	int Total;
	float Avg;
};

void main()
{
	int i;
	struct Student S;

	printf("\n\nEnter Student Roll : ");
	scanf("%d",&S.Roll);
	printf("Enter Student Name : ");
	scanf("%s",&S.Name);
	S.Total = 0;
	for(i=0;i<3;i++)
	{
		printf("Enter Marks %d : ",i+1);
		scanf("%d",&S.Marks[i]);
		S.Total = S.Total + S.Marks[i];
	}
	S.Avg = S.Total / 3;

	printf("\nRoll : %d",S.Roll);
	printf("\nName : %s",S.Name);
	printf("\nTotal : %d",S.Total);
	printf("\nAverage : %f",S.Avg);
}
```

```
Enter Student Roll : 1
Enter Student Name : Viraj
Enter Marks 1 : 10
Enter Marks 2 : 20
Enter Marks 3 : 30

Roll : 1
Name : Viraj
Total : 60
Average : 20.000000
```

Define a structure called Result for students. Structure will have members like Roll number, marks for three subjects and total of three subjects. Write a program to enter data for 5 students and display the merit list of students. Use Array of Structures. For example, if Roll No and marks of three subjects of each student are entered through the keyboard, the output should look like the following:

```
Merit list -->

Roll_No  Sub1  Sub2  Sub3 Total
    3     90    89    98   277
    4     89    78    98   265
    2     90    78    89   257
    5     89    78    90   257
    1     45    67    89   201
```

Example

```
#include<stdio.h>
struct result
{
    int rollno;
    int sub[3];
    int total;
};

void main()
{
    int i, n, j;
    struct result st[20], temp;
    printf("Enter number of students data you want to enter:\n");
    scanf("%d",&n);
    for(i=0;i < n;i++)
    {
        printf("Enter Roll No of student %d\n",(i+1));
        scanf("%d",&st[i].rollno);
        printf("Enter marks for 3 subjects of student %d\n",(i+1));
        scanf("%d%d%d",&st[i].sub[0],&st[i].sub[1],&st[i].sub[2]);
        st[i].total = (st[i].sub[0]+st[i].sub[1]+st[i].sub[2]);
    }
```

Example

```
    for(i=0;i < (n-1);i++)
    {
        for(j=0;j < (n-i-1);j++)
        {
            if(st[j].total < st[j+1].total)
            {
                temp = st[j];
                st[j] = st[j+1];
                st[j+1] = temp;
            }
        }
    }
    printf("\nMerit List -->\n");
    printf("Roll No\tSub1\tSub2\tSub3\tTotal\n");

    for(i=0; i < n;i++)
    {
        printf("\n%d\t%d\t%d\t%d\t%d",st[i].rollno,st[i].sub[0],st[i].sub[1],st[i].sub[2],st[i].total);
    }
}
```

Example

```
Enter number of students data you want to enter:
5
Enter Roll No of student 1
1
Enter marks for 3 subjects of student 1
45
67
89
Enter Roll No of student 2
2
Enter marks for 3 subjects of student 2
90
78
89
Enter Roll No of student 3
3
Enter marks for 3 subjects of student 3
90
89
98
Enter Roll No of student 4
4
Enter marks for 3 subjects of student 4
89
78
98
Enter Roll No of student 5
5
Enter marks for 3 subjects of student 5
89
78
90
```

```
Merit List -->
Roll_No Sub1    Sub2    Sub3    Total

3       90      89      98      277
4       89      78      98      265
2       90      78      89      257
5       89      78      90      257
1       45      67      89      201
```

Let’s Practice

1. Define a structure ticket having three members, name of passenger, number of tickets booked and ticket price per ticket. Enter the data of five passengers and print name and total charge of all tickets booked per passenger.
2. Define a Structure player having data members: name, country, runs, no of matches and age. Write a program to enter data for 10 cricketers and print the details of all the players having age less than 31 years.
3. Define a structure for book having members Title, Price, Year of publication and Author. Input data for 10 books and print all the books which is older than a particular year entered from user.
4. Define a Structure named Census with the following three data members: name of the city, population of the city in long integer and number of literate person. Enter the details of 3 cities using array of structure and display total number of illiterate person in each city.
5. Define structure for Mobile having members Model, Price, Company, Year of Manufacture. Input data for 5 mobiles and print all the mobile details having price less than 10000.

Let’s Practice

6. Define a structure name country having members like country name, population, and national language. Input data of 3 countries, using arrays of structure and find country having highest population.

7. Define structure named Student with members: int id, mark1, mark2. Write a C program that enters data for 5 students using structure and print individual total marks and average of marks of each student.

8. Define a structure called cricket that will describe: player name, team average and batting average. Using cricket, declare an array player with 50 elements and write a program to read the information about all the 50 players and print team-wise list containing names of players with their batting average.

9 Define a structure that describes a student having members roll no, name, mark1, mark2, mark3. Write a program for 10 students to print the total of marks of each student and the highest of them.

Structure within Structure

- Structure within structure means **nesting of structures**.

```
struct salary
{
    char name[10];
    char department[10];
    int basic_pay;
    int DA;
    int HRA;
    int CA;
}employee;
```

```
struct salary
{
    char name[10];
    char department[10];
    int basic_pay;

    struct benefit
    {
        int DA;
        int HRA;
        int CA;
    }allowance;

}employee;
```

NOTE: The salary structure contains a member named allowance, which itself is a structure with three members.

Structure within Structure

An inner-most member in a nested structure can be accessed by chaining all the concerned structure variables(**from outer-most to inner-most**) with the member using dot operator.

```
employee.allowance.DA
employee.allowance.HRA
employee.allowance.CA
```

```
employee.allowance  //INVALID: actual member is missing
employee.HRA  //INVALID: inner structure variable is missing
```

Structure within Structure

An inner structure can have more than one variable.

```
struct salary
{
    char name[10];
    char department[10];
    int basic_pay;

    struct benefit
    {
        int DA;
        int HRA;
        int CA;
    }allowance,arrears;

}employee[100];
```

Accessing inner member:

```
employee[1].allowance.DA
employee[1].arrears.DA
```

Structure within Structure

We can also use tag names to define inner structure.

```
struct benefit
{
    int DA;
    int HRA;
    int CA;
};

struct salary
{
char name[10];
char department[10];
int basic_pay;

struct benefit allowance;
struct benefit arrears;
}employee[100];
```

It is also permissible to nest more than one type of structures.

```
struct personal_record
{
    struct name_part name;
    struct addr_part address;
    struct date date_of_birth;
    ..........
    ..........
}person1;
```

Example: Structure within Structure

Write a program to read and display information of salary of an employee using Structure within a Structure. Outer structure contains members like name of employee, designation, department name, basic pay and inner structure contains dearness allowance, house rent allowance and city allowance. Calculate the total salary of one employee.

Example: Structure within Structure

```c
#include<stdio.h>
struct employee
{
    char name[20];
    char des[20];
    char dept_name[20];
    int basic_pay;

    struct payment
    {
        int da;
        int hra;
        int ca;
    }pay;
}emp;
```

```c
void main()
{
    int total;
    printf("***Enter Employee Data**\n\n");

    printf("Enter name of employee:");
    scanf("%s",emp.name);
    printf("Enter designation:");
    scanf("%s",emp.des);
    printf("Enter department name:");
    scanf("%s",emp.dept_name);
    printf("Enter basic pay:");
    scanf("%d",&emp.basic_pay);
```

Example: Structure within Structure

```
printf("Enter dearness allowance:");
scanf("%d",&emp.pay.da);
printf("Enter house_rent allowance:");
scanf("%d",&emp.pay.hra);
printf("Enter city_allowance.:");
scanf("%d",&emp.pay.ca);

printf("\n***Displaying Employee Data**\n\n");

printf("Name:%s \ndesignation:%s \ndepartment name: %s \nbasic pay:%d\n",emp.name,emp.des,emp.dept_name,emp.basic_pay);
printf("dearness allowance:%d \nhouse_rent allowance:%d \ncity_allowance:%d",emp.pay.da,emp.pay.hra,emp.pay.ca);
total=emp.basic_pay+emp.pay.da+emp.pay.hra+emp.pay.ca;
printf("\nTotal Salary=%d",total);
}
```

Example: Structure within Structure

```
***Enter Employee Data**

Enter name of employee:Nishat
Enter designation:Professor
Enter department name:CSPIT-IT
Enter basic pay:50000
Enter dearness allowance:1000
Enter house_rent allowance:1000
Enter city_allowance.:1000

***Displaying Employee Data**

Name:Nishat
designation:Professor
department name: CSPIT-IT
basic pay:50000
dearness allowance:1000
house_rent allowance:1000
city_allowance:1000
Total Salary=53000
```

Let’s Practice

Write a C program to create structure named SALARY having data members employee name, basic salary, gross salary and one inner structure of ALLOWANCE having data members DA and HRA. Take name and basic salary as input from user for one employee and calculate DA, HRA and gross salary. Display employee name and gross salary on screen.

NOTE: DA = 136% of basic salary

HRA = 10% of basic salary

Gross salary = basic salary + DA + HRA

Practical 10.3

Write a C program for nested structure to display employee details such as, Age, Name, Address, Salary.

Structure as function arguments

- 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.

Structure as function arguments

Passing Structure To Function can be done in below 3 ways:

- Passing Structure Members as arguments to Function
- Passing structure to a function **by value**
  (Passing Structure Variable as Argument to a Function)
- Passing structure to a function **by address**
  (Passing Structure Pointers as Argument to a Function)

Passing Structure Members as arguments to Function

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

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

void print_struct(char name[], int roll_no, int marks);

void main()
{
    struct student stu = {"Tim", 1, 78};
    print_struct(stu.name, stu.roll_no, stu.marks);
}

void print_struct(char name[], int roll_no, int marks)
{
    printf("Name: %s\n", name);
    printf("Roll no: %d\n", roll_no);
    printf("Marks: %d\n", marks);
}
```

```
Name: Tim
Roll no: 1
Marks: 78
```

Passing structure to a function by value

- if a structure contains two-three members then we can easily pass them to function individually
- But passing more members individually is a tiresome and error-prone process.
- So in such cases we can pass structure variable itself.

Passing structure to a function by value

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

void print_struct(struct student stu);

void main()
{
    struct student stu = {"George", 10, 69};
    print_struct(stu);

    //printf("New name: %s", stu.name);

}

void print_struct(struct student stu)
{
    printf("Name: %s\n", stu.name);
    printf("Roll no: %d\n", stu.roll_no);
    printf("Marks: %d\n", stu.marks);

    //printf("\nChanging name ... \n");
    //strcpy(stu.name, "Jack");
}
```

```
Name: George
Roll no: 10
Marks: 69
```

Passing structure to a function by address

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.

- 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.

- Additional memory is required to save every copy of the structure.

Passing structure to a function by address

```c
struct employee
{
    char name[20];
    int age;
    char doj[10]; // date of joining
    char designation[20];
};

void print_struct(struct employee *ptr);

void main()
{
    struct employee dev = {"Jane", 25, "25/2/2015", "Developer"};
    print_struct(&dev);

    //printf("New name: %s", dev.name);

}

void print_struct(struct employee *ptr)
{
    printf("Name: %s\n", ptr->name);   //(*ptr).name
    printf("Age: %d\n", ptr->age);     //(*ptr).age
    printf("Date of joining: %s\n", ptr->doj);       //(*ptr).doj
    printf("Designation: %s\n", ptr->designation);   //(*ptr).designation

    //printf("\nChanging name ... \n");
    //strcpy(ptr->name, "Jack");
}
```

```
Name: Jane
Age: 25
Date of joining: 25/2/2015
Designation: Developer
```

Return struct from a function

```c
struct student
{
    char name[50];
    int age;
};

struct student getInformation();

void main()
{
    struct student s;

    s = getInformation();

    printf("\nDisplaying information\n");
    printf("Name: %s", s.name);
    printf("\nRoll: %d", s.age);
}

struct student getInformation()
{
  struct student s1;

  printf("Enter name: ");
  scanf ("%s", s1.name);

  printf("Enter age: ");
  scanf("%d", &s1.age);

  return s1;
}
```

```
Enter name: Nishat
Enter age: 25

Displaying information
Name: Nishat
Roll: 25
```

Example

Define a structure named Date that contains three member's day, month and Year. Write a program that compares two given dates. If the dates are equal, then display message as "Equal" otherwise "Unequal". Write a function Check_Date to check whether the entered date is proper or not. The date is proper if day is between 1 and 31, month is between 1 and 12 and year is between 1000 and 9999. (Structures & Functions)

```c
#include<stdio.h>
struct date
{
    int day;
    int month;
    int year;
};

void check_date(struct date d);

void main()
{
    struct date d1,d2;

    printf("Enter first date(dd/mm/yyyy):");
    scanf("%d%d%d",&d1.day,&d1.month,&d1.year);
    check_date(d1);

    printf("\nEnter second date(dd/mm/yyyy):");
    scanf("%d%d%d",&d2.day,&d2.month,&d2.year);
    check_date(d2);

    if((d1.day==d2.day)&&(d1.month==d2.month)&&(d1.year==d2.year))
        printf("\nEQUAL");
    else
        printf("\nUNEQUAL");
}
```

```c
void check_date(struct date d)
{
   if(d.year>=1000 && d.year<=9999)
    {
        if(d.month>=1 && d.month<=12)
        {
            if((d.day>=1 && d.day<=31) && (d.month==1 || d.month==3 || d.month==5 || d.month==7 || d.month==8 || d.month==10 || d.month==12))
                printf("Date is valid.\n");
            else if((d.day>=1 && d.day<=30) && (d.month==4 || d.month==6 || d.month==9 || d.month==11))
                printf("Date is valid.\n");
            else if((d.day>=1 && d.day<=28) && (d.month==2))
                printf("Date is valid.\n");
            else if(d.day==29 && d.month==2 && (d.year%400==0 ||(d.year%4==0 && d.year%100!=0)))
                printf("Date is valid.\n");
            else
                printf("Day is invalid.\n");
        }
        else
        {
            printf("Month is not valid.\n");
        }
    }
    else
    {
        printf("Year is not valid.\n");
    }
}
```

```
Enter first date(dd/mm/yyyy):
02
02
1995
Date is valid.

Enter second date(dd/mm/yyyy):
02
02
1995
Date is valid.

EQUAL
```

Let’s Practice

Define a structure that describes a hotel. It should have member that included name, address, grade, average room charge, and number of rooms. Write a function to print out all hotels of a given grade in order of charges. Also print out hotels with room charges less than given values.

Practical 10.1

Write a C program to create a structure of Book Detail and display the details of the book in appropriate format by passing structure as function argument.

1. Using dot notation:

VariableName.MemberName

2. Using indirection notation:

(*PointerName).MemberName

3. Using arrow notation:

PointerName->MemberName

- **typedef** is a keyword used in C language to assign **alternative names** to existing datatypes.

- Its mostly used with user defined datatypes, when names of the datatypes become slightly complicated to use in programs

Syntax:

```
typedef <existing_name> <alias_name>
```

```c
struct student
{
    char name[20];
    int age;
};
struct student s1;
```

Using typedef

Method 1

```c
struct student
{
    char name[20];
    int age;
};
typedef struct student stud;
stud s1;
```

Method 2

```c
typedef struct student
{
    char name[20];
    int age;
}stud;
stud s1,s2;
```

NOTE: In Method 2 stud is not variable. It is the type definition name.

- **Unions** are a concept borrowed from structures .Follow the same syntax as structures
- Union is also a **derived type** as structure.
- However major distinction between them is in terms of **storage**.
- In structure, each member has its own storage location
- In union all the members use the same location
- This implies that, although a union may contain many members of different types, it can handle only one member at a time.
- Compiler allocates a piece of storage that is large enough to hold the largest variable type in union

Example:

Defining a Union

```
union item
{
    int m;
    float x;
    char c;
};
```

Declaring Union Variables

Method 1

```
union item
{
    int m;
    float x;
    char c;
}i1,i2,i3;

void main()
{

}
```

Method 2

```
union item
{
    int m;
    float x;
    char c;
};

void main()
{
    union item i1,i2,i3;
}
```

Access member

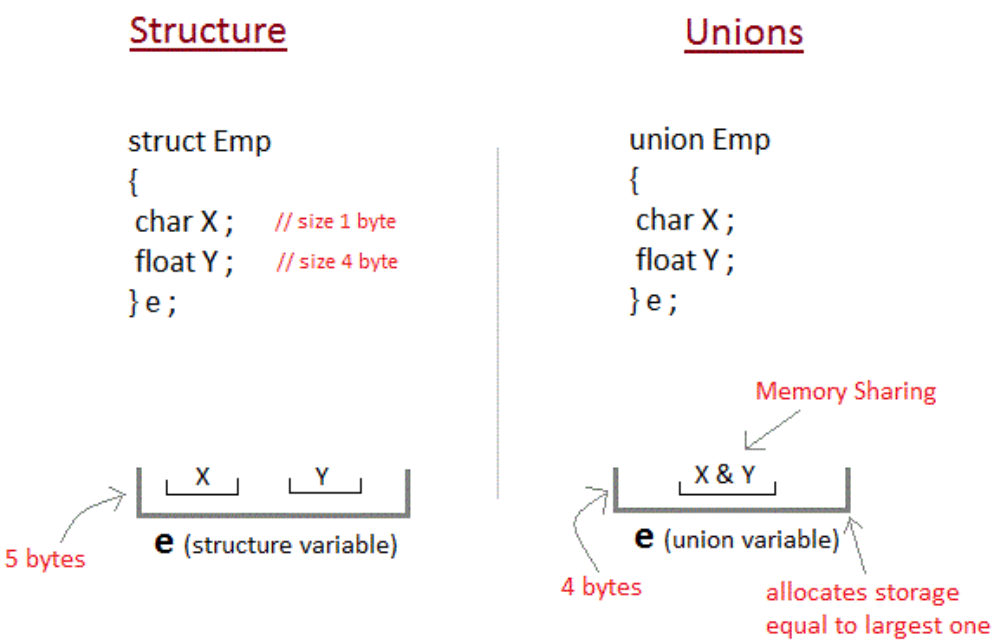
```
code.m
//print value
code.x
//print value
code.c
//print value
```

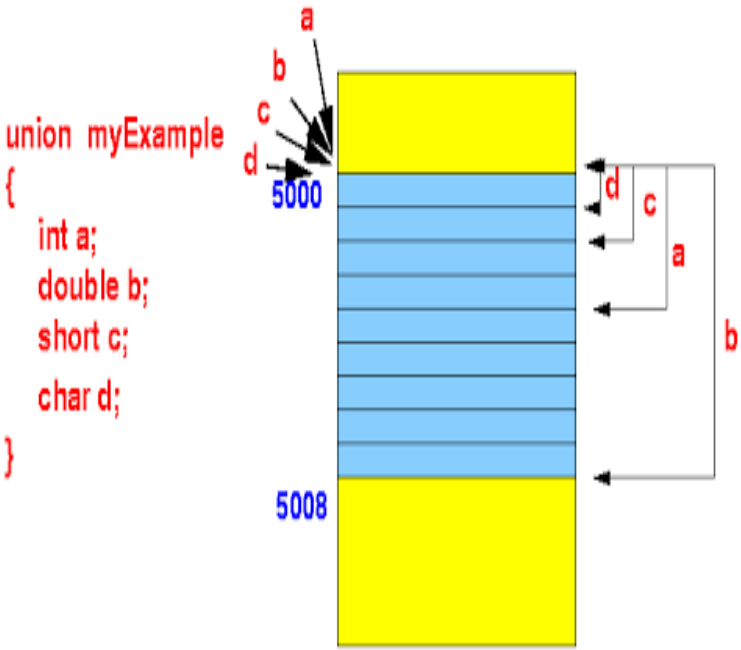
```
code.m=379;
code.x=7859.36;
printf("%d",code.m);
```

Erroneous Output

There is difference in memory allocation between union and structure .

- The amount of memory required to store a structure variables is the sum of memory size of all members.
- But, the memory required to store a union variable is the memory required for largest element of an union.





Union

```c
union Address
{
    char name[50];
    char street[100];
};

void main()
{
    union Address std = {"Nishat", "75th Street"};
    printf("\nname: %s",std.name);
    printf("\nstreet: %s",std.street);
}
```

warning: excess elements in union initializer

```
name: Nishat
street: Nishat
```

```c
union Date
{
int dd,mm,yyyy;
};

void main()
{
    union Date D;

    D.dd=15;
    printf("After making dd = 15:\ndd = %d\nmm = %d\nyyyy=%d\n\n",D.dd,D.mm,D.yyyy);

    D.yyyy = 2002;
    printf("After making yyyy = 2002:\ndd = %d\nmm = %d\nyyyy=%d\n",D.dd,D.mm,D.yyyy);
}
```

```
After making dd = 15:
dd = 15
mm = 15
yyyy=15

After making yyyy = 2002:
dd = 2002
mm = 2002
yyyy=2002
```

Union

```c
#include <stdio.h>

union job
{
    char name[32];
    float salary;
}u;

void main()
{
    printf("Enter name:\n");
    scanf("%s",&u.name);

    printf("Enter salary: \n");
    scanf("%f",&u.salary);

    printf("\nName :%s\n",u.name);
    printf("salary: %.1f",u.salary);
}
```

```
Enter name:
Nishat
Enter salary:
75000

Name :
salary: 75000.0
```

```c
#include <stdio.h>

union job
{
    char name[32];
    float salary;
}u;

void main()
{
    printf("Enter name:\n");
    scanf("%s",&u.name);
    printf("Name :%s\n",u.name);


    printf("\nEnter salary: \n");
    scanf("%f",&u.salary);
    printf("salary: %.1f",u.salary);
}
```

```
Enter name:
Nishat
Name :Nishat

Enter salary:
75000
salary: 75000.0
```

OBSERVE OUTPUT

Size of Union

```c
union student
{
    char name[20];    //1 Byte each = 20 Bytes
    int roll_no;      //4 Bytes
    float marks[2];   //4 Byte each = 8 Bytes
};

void main()
{
    union student std[2];
    printf("\n union size : %d Bytes\n",sizeof(union student));
}
```

```
union size : 20 Bytes
```

- Union can be initialized only with a value of the same type as the first union member

	structure	union
Keyword	"struct" is used to define the structure.	"union" is used to define union.
Size	When a variable is associated with the structure, compiler allocates memory for each member. The size of structure is greater than or equal to the sum of the sizes of it's members.	When a variable is associated with the union, compiler allocates memory by considering the size of the largest member. The size of union is the size of the largest member.
Memory	Each member is assigned a unique storage location.	Memory allocated is shared by individual members.
Value altering	Altering value of a member will not affect other members.	Altering value of a member will affect value of other members.
Accessing members	Individual members can be accessed at a time.	Only one member can be accessed at a time.
Initialization of members	Many members of a structure can be initialized at a time.	Only the first member of a union can be initialized.

Create a Union called library to hold accession number, title of the book, author name, price of the book and flag indicating whether the book is issued or not. (flag = 1 if the book is issued, flag = 0 otherwise). Write a program to enter data of one book and display the data.

- So far we have been using integer fields of size 16/32 bits to store data
- There are occasions where data items require much less than 16/32 bit space.
- In such cases, we **waste memory space**.
- C permits us to use small bit fields to hold data items
- we can specify size (in bits) of structure and union members.
- The idea is to **use memory efficiently** when we know that the value of a field or group of fields will never exceed a limit or is within a small range.

Need for Bit Fields in C

- Used to reduce memory consumption.
- Easy to implement.
- Provides flexibility to the code

Definition of Bit Fields

The variables defined with a predefined width are called **bit fields**.

Declaration of Bit Fields in C

```
struct tag-name
{
    data_type variable_name1 : size_in_bits;
    data_type variable_name2 : size_in_bits;
};
```

consider the following declaration of date without the use of bit fields.

```
struct date
{
    unsigned int d;
    unsigned int m;
    unsigned int y;
};

void main()
{
    printf("Size = %d bytes",sizeof(struct date));
}
```

```
Size = 12 bytes
```

Since we know that the value of d is always from 1 to 31, the value of m is from 1 to 12, we can optimize the space using bit fields.

consider the following declaration of date with the use of bit fields.

```
struct date
{
    // d has value between 1 and 31(11111), so 5 bits are sufficient
    unsigned int d : 5;

    // m has value between 1 and 12(1100), so 4 bits are sufficient
    unsigned int m : 4;

    unsigned int y;
};

void main()
{
    printf("Size = %d bytes", sizeof(struct date));
}
```

```
Size = 8 bytes
```

NOTE:

- We cannot use scanf to read values into bit fields.
- We can neither use pointer to access the bit fields
- Bit fields cannot be arrayed

End of Unit-10