Programming in C

I semester, BCA

Ambition Guru College

Compiled by :
Ankit Bhattarai

Unit 6

(7 hrs.)

Structures and Unions

Concept of Structure, Initializing, accessing member of structure, Array of structure, Structures and Function, Pointer to structure, Union, Different between union and structure

What is a Structure?

- Structure is a collection of different or similar data types variable under a common variable name. It is used for handling a group of logically related data items.
    - Examples:
        - Student name, roll number, and marks
        - Real part and complex part of a complex number
- Helps in organizing complex data in a more meaningful way
- The individual structure elements are called members.

Defining a Structure


```
struct structure_name

          {
           member 1;
           member 2;
           :
           member m;

          };
```

- struct is the required C keyword
- structure_name is the name of the structure
- member 1, member 2, … are individual member declarations.

Example:

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

Here, a derived type struct student is defined. Student is a structure name and name, roll, marks, are called structure elements or members. Each of these members belongs to different data types.

Contd.

- 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
- Once a structure has been defined, the individual structure-type variables can be declared as:

```
struct structure_name var_1, var_2, …, var_n;
```

Example

- A structure definition

```
struct student {
        char name[30];
        int  roll_number;
        int  total_marks;
        char dob[10];
             };
```

- Defining structure variables:

```
struct student  a1, a2, a3;
```

A new data-type

A Compact Form

- It is possible to combine the declaration of the structure with that of the structure variables:

```
struct tag {
        member 1;
        member 2;
        :
        member m;

        }  var_1, var_2,…, var_n;
```

- Declares three variables of type struct tag
- In this form, tag is optional

Accessing a Structure

- The members of a structure are processed individually, as separate entities
    - Each member is a separate variable.
- A structure member can be accessed by writing: variable.member

  where variable refers to the name of a structure-type variable, and member refers to the name of a member within the structure
- Examples: a1.name, a2.name, a1.roll_number, a3.dob

Accessing a Structure

- To access any member of a structure, we use the dot operator (.) .

Syntax:

`structure_variable.member`

Here structure variable refers to the name of structure type variable and member refers to name of member within structure.

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

int main()
{
  struct student st;
  strcpy(st.name,"Ram");
  st.roll=15;
  st.marks=85.5;
  printf("Name of student is %s",st.name);
  printf("\nRoll number= %d",st.roll);
  printf("\nMarks obtained=%f",st.marks);
  getch();
  return 0;
}
```

Output

Name of student is Ram
Roll number=15
Marks Obtained=85.5

Question 1

Create a structure called student having name, roll, age and address. Input the record from the user and display it.

```c
#include <stdio.h>

struct student {
    char name[50];
    int roll;
    int age;
    char address[100];
};

int main() {
    struct student s;

    printf("Enter name: ");
    gets(s.name);

    printf("Enter roll number: ");
    scanf("%d", &s.roll);

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

    printf("Enter address: ");
    gets(s.address);

    printf("Name    : %s\n", s.name);
    printf("Roll No : %d\n", s.roll);
    printf("Age     : %d\n", s.age);
    printf("Address : %s\n", s.address);

    return 0;
}
```

Question 2

Create a structure called customer having name, address, balance and accountnumber. Input a record and display its information.

```c
#include <stdio.h>

struct customer {
    char name[50];
    char address[100];
    float balance;
    int accountnumber;
};

int main() {
    struct customer c;

    printf("Enter customer name: ");
    gets(c.name);

    printf("Enter address: ");
    gets(c.address);

    printf("Enter account number: ");
    scanf("%d", &c.accountnumber);

    printf("Enter account balance: ");
    scanf("%f", &c.balance);

    printf("Name           : %s\n", c.name);
    printf("Address        : %s\n", c.address);
    printf("Account Number: %d\n", c.accounnumber);
    printf("Balance        : %.2f\n", c.balance);

    return 0;
}
```

Array of structures

Arrays of Structures

- An array of structures is a collection of structure variables stored under one name, where each element is a structure.
- It allows you to store and manage multiple records of the same structure type (e.g., multiple students, employees, etc.).

Syntax:

```
struct StructureName {
    dataType member1;
    dataType member2;
    ...
};

struct StructureName variableName[size];
```

Arrays of Structures

- Once a structure has been defined, we can declare an array of structures

```
struct student s[50];
```

  - The individual members can be accessed as:

```
s[i].name

s[5].roll_number
```

How to declare and initialize array of structure variables ?

How to declare and initialize array of structure variables?

- Suppose we want declare a structure to store 5 employee details. Array of structure to store 5 employee details is can be declared as follows:

```
struct employee
{
    char name[20];
    int id;
    float salary;
};

int main()
{
    struct employee e[5];
}
```

How to declare and initialize array of structure variables?

- Structure array initialization follows same syntax as we initialize single structure variable. The only difference is here we can initialize more than one structure variable at a time.

```
struct employee e[5] =
{
    { "Ramesh", 12,15000.35},
    { "Hari", 15, 25000.5 },
    { "Gopal", 17, 30000.25 },
    { "Sita", 2, 45000.56 },
    { "Roshan", 9, 32000.25}
};
```

Array of structures

Example:

Create a structure named student having name, roll and age. Take the input of 3 students and display the information.

```c
#include <stdio.h>

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

int main() {
    struct student s[3];
    int i;

    for (i = 0; i < 3; i++) {
        printf("Name: ");
        gets(s[i].name);
        printf("Roll: ");
        scanf("%d", &s[i].roll);
        printf("Age: ");
        scanf("%d", &s[i].age);
    }

    printf("\n--- Student Records ---\n");
    for (i = 0; i < 3; i++) {
        printf("Name: %s\n", s[i].name);
        printf("Roll: %d\n", s[i].roll);
        printf("Age: %d\n\n", s[i].age);
    }

    return 0;
}
```

Create a structure named employee having name, id and salary.
Take the input of n employees from the user and display the information.

Create a structure named employee having name, id and salary.
Take the input of n employees from the user and display the information.


```
#include <stdio.h>

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

int main()
{
int n, i;
printf("Enter the value of n");
scanf("%d", &n);

struct employee e[n];

for (i = 0; i < n; i++) {
        printf("Name: ");
        gets(s[i].name);
        printf("id: ");
        scanf("%d", &s[i].id);
        printf("Salary: ");
        scanf("%f", &s[i].salary);
    }

    for (i = 0; i < n; i++) {
        printf("Name: %s\n", s[i].name);
        printf("Id: %d\n", s[i].id);
        printf("Salary: %f\n\n", s[i].salary);
    }

    return 0;
}
```

Question:

Create a structure called customer having name, address, accountnumber and balance. Input n records and

- Display information of customer whose balance is greater than minimum balance 10000

Output:

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

struct customer {
    char name[50];
    char address[100];
    int accountNumber;
    float balance;
};

int main() {
    int n, i, searchAcc, found = 0;

    printf("Enter number of customers: ");
    scanf("%d", &n);

    struct customer c[n];

    for (i = 0; i < n; i++) {
        printf("Name: ");
        gets(c[i].name);

        printf("Address: ");
        gets(c[i].address);

        printf("Account Number: ");
        scanf("%d", &c[i].accountNumber);

        printf("Balance: ");
        scanf("%f", &c[i].balance);
    }
```



```
printf("\n--- Customers with balance > 10000 ---\n");
  for (i = 0; i < n; i++) {
    if (c[i].balance > 10000) {
      printf("Name: %s\n", c[i].name);
      printf("Account Number: %d\n", c[i].accountNumber);
      printf("Balance: %.2f\n\n", c[i].balance);
    }
  }


  return 0;
}
```

Arrays within Structures

- A structure member can be an array

```
struct  student
{
      char  name[30];
      int  roll_number;
      int  marks[5];
      char  dob[10];
}  a1, a2, a3;
```

- The array element within the structure can be accessed as:

```
a1.marks[2], a1.dob[3],…
```

Arrays within Structures: Example


```
#include <stdio.h>

struct Student {
    char name[50];
    int marks[5];
};

int main() {
    struct Student s1 = {"John", {85, 90, 78, 92, 88}};

    printf("Name: %s\n", s1.name);
    printf("Marks: ");
    for(int i = 0; i < 5; i++) {
        printf("%d ", s1.marks[i]);
    }
    printf("\n");

    return 0;
}
```

Array of structures

Create a structure called employee having name, address, salary and age. Input n records of employee and display information of employee whose address is Kathmandu.

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
struct employee
{
char name[20];
char address[20];
float salary;
int age;
};

int main()
{
  struct employee e[100];
  int i,n;
  printf("Enter how many records you want to enter");
  scanf("%d",&n);
  for(i=0;i<n;i++)
  {
    printf("Enter the records of employee %d\n",i+1);
    printf("Enter the name");
    gets(e[i].name);
    printf("Enter the address");
    gets(e[i].address);
    printf("Enter the salary");
    scanf("%f",&e[i].salary);
    printf("Enter the age");
    scanf("%d",&e[i].age);
}
```

Array of structures

Create a structure called employee having name, address, salary and age. Input n records of employee and display information of employee whose address is Kathmandu.

```
printf("\nName \tAddress \tSalary \tAge");

for(i=0;i<n;i++)
{
  if(strcmp(e[i].address,"kathmandu")==0)
  {
   printf("\n%s\t%s\t%f\t%d",e[i].name,e[i].address,
   e[i].salary,e[i].age);
  }
}
getch();
return 0;
}
```

Create a structure called book, name, price, author, and published date in day, month, and year. Write a program to read 100 books information from the user and displays those records having price greater than 250.

Solution:


```
#include<stdio.h>
#include<conio.h>
struct date
{
  int dd;
  int mm;
  int yy;
};

struct book
{
  char name[20];
  float price;
  char author[20];
  struct date d;
};
```

```
int main()
{
int i;
struct book b[100];
for(i=0;i<100;i++)
{
printf("Enter the information of book %d\n",i+1);
printf("Enter the book name");
gets(b[i].name);
printf("Enter the author name");
gets(b[i].author);
printf("Enter the book price");
scanf("%f",&b[i].price);
printf("Enter book published year");
scanf("%d",&b[i].d.yy);
printf("Enter book published month");
scanf("%d",&b[i].d.mm);
printf("Enter book published date");
scanf("%d",&b[i].d.dd);
}
```

Solution:

```
printf("\nInformation of Books having price greater than 250 are\n");
printf("\nName\tPrice\tAuthor\tPublished date");

for(i=0;i<100;i++)
{
    if(b[i].price>250)
    {
printf("\n%s\t%f\t%s\t%d\t%d\t%d",b[i].name,b[i].price,b[i].author,b[i
].d.dd,b[i].d.mm, b[i].d.yy);
     }
}
getch();
return 0;
}
```

Nested Structure

Nested Structure

- A **nested structure** means a structure **within another structure**.
- It allows grouping related data more logically and hierarchically.

Syntax:

```
struct Inner {
    // members
};

struct Outer {
    // other members
    struct Inner innerVar;  // Nested structure
};
```

Nested Structure

- Example:

```
#include<stdio.h>
#include<conio.h>

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

struct employee
{
char name[20];
int id;
struct date dob;
float salary;
}e;
```

Nested Structure

- Example:

```
int main()
{
printf("Enter the name of employee");
scanf("%s",e.name);

printf("Enter ID of employee");
scanf("%d",&e.id);

printf("Enter the year of birthday\n");
scanf("%d",&e.dob.year);

printf("Enter the month of birthday\n");
scanf("%d",&e.dob.month);

printf("Enter the day of birthday\n");
scanf("%d",&e.dob.day);

printf("Enter the salary of employee\n");
scanf("%f",&e.salary);

printf("Name\tId\tDay\tMonth\tYear\tsalary\n");
printf("%s\t%d\t%d\t%d\t%d\t%f",e.name,e.id,e.dob.day,e
.dob.month,e.dob.year,e.salary);
getch();
return 0;
}
```

Pointer to Structure

Pointer to Structure

- Pointers can also be used pointing to a structure.
- We can define pointers to structures in the following way

```
struct structure_name *pointer_name;
```

- We can store the address of a structure variable in following way:

```
pointer_name = &structure_variable name;
```

- To access the members of a structure using a pointer to that structure, you must use the → operator as follows:

```
pointer_name->member;
```

Pointer to Structure

- Example:

```
#include<stdio.h>
#include<conio.h>
struct book
{
char name[30];
int pages;
float price;
};

int main()
{
struct book b={"C programming",300,550.30};
struct book *ptr;
ptr=&b;
printf("\nBook name\tNo.of pages\tprice\n");
printf("\n%s\t%d\t%f",ptr->name,ptr->pages,ptr->price);
getch();
return 0;
}
```

Structure and Function

In C , structure can be passed to functions by two methods.

1) Passing by value (Passing actual value as argument)

2) Passing by reference (Passing address of an argument)

Parameter Passing in a Function

- Structure variables can be passed as parameters like any other variables. Only the values will be copied during function invocation.

```
void swap (struct complex a, struct complex b)
{
    struct complex tmp;

    tmp=a;
    a=b;
    b=tmp;
}
```

Structure and Function : Passing by value

- This methods involves passing a copy of the entire structure to the called function. Any changes to structure members within the function are not reflected in original structure in the calling function.

```
#include<stdio.h>
#include<conio.h>
struct employee
{
char name[50];
float salary;
};
void display(struct employee e);
void addbonus(struct employee ee);

int main()
{
struct employee emp;
printf("Enter the name of employee");
scanf("%s",emp.name);
printf("Enter the salary of employee");
scanf("%f",&emp.salary);
addbonus(emp);
display(emp);
getch();
}
```

Structure and Function : Passing by value

- This methods involves passing a copy of the entire structure to the called function. Any changes to structure members within the function are not reflected in original structure in the calling function.

```
.void addbonus(struct employee ee)
 {
 ee.salary=ee.salary+5000;
 }

 void display(struct employee e)
 {
 printf("\nInformation of Employee");
 printf("\nName:%s",e.name);
 printf("\nSalary:%f",e.salary);
 }
```

Output

Enter the name of employee: Ramesh
Enter the salary of employee: 25000
Information of employee
Name: Ramesh
Salary: 25000

Structure and Function : Passing by reference

- In this method the address location of the structure is passed to the called function. The function can access indirectly the entire structure and work on it. Any changes made in structure variable in function definition reflects in original structure variable in calling function.

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

void display(struct employee *e);
void addbonus(struct employee *ee);

int main()
{
struct employee emp;
printf("Enter the name of employee");
scanf("%s",emp.name);
printf("Enter the salary of employee");
scanf("%f",&emp.salary);
addbonus(&emp);
display(&emp);
getch();
}
```

Structure and Function : Passing by reference

- In this method the address location of the structure is passed to the called function. The function can access indirectly the entire structure and work on it. Any changes made in structure variable in function definition reflects in original structure variable in calling function.

```
void addbonus(struct employee *ee)
{
ee->salary=ee->salary+5000;
}

void display(struct employee *e)
{
printf("\nInformation of Employee");
printf("\nName:%s",e->name);
printf("\nSalary:%f",e->salary);
}
```

Output:

Enter the name of employee: Ramesh
Enter the salary of employee: 25000
Information of employee
Name: Ramesh
Salary: 30000

Returning structures

- It is also possible to return structure values from a function. The return data type of the function should be as same as the data type of the structure itself

```
struct complex add(struct complex a, struct complex b)
{
    struct complex tmp;

    tmp.real = a.real + b.real;
    tmp.imag = a.imag + b.imag;
    return(tmp);
}
```

Direct arithmetic operations are not possible with structure variables

Write a program in C to add two complex numbers using structure and function.


```
#include <stdio.h>

struct complex {
    float real;
    float imag;
};

struct complex add(struct complex a, struct complex b) {
    struct complex tmp;
    tmp.real = a.real + b.real;
    tmp.imag = a.imag + b.imag;
    return tmp;
}
```

Write a program in C to add two complex numbers using structure and function.


```
int main() {
    struct complex num1, num2, result;

    printf("Enter real and imaginary part of first complex number:\n");
    printf("Real: ");
    scanf("%f", &num1.real);
    printf("Imaginary: ");
    scanf("%f", &num1.imag);

    printf("\nEnter real and imaginary part of second complex number:\n");
    printf("Real: ");
    scanf("%f", &num2.real);
    printf("Imaginary: ");
    scanf("%f", &num2.imag);

    result = add(num1, num2);

    printf("\nSum of complex numbers = %.2f + %.2fi\n", result.real, result.imag);

    return 0;
}
```

Write a program in C to add two complex numbers using structure and function.

Output:

```
Enter real and imaginary part of first complex number:
Real: 3.2
Imaginary: 4.5

Enter real and imaginary part of second complex number:
Real: 1.8
Imaginary: 2.3

Sum of complex numbers = 5.00 + 6.80i
```

Union

Union

- A union is a user-defined data type in C that allows storing different data types in the same memory location. However, only one member can contain a value at any given time—all members share the same memory space.

Defining a Union

```
union union_name
                    {
         member 1;
         member 2;
         :
         member m;
        };
```

union is the C keyword
union_name is the name of the structure
member 1, member 2, … are individual member declarations.

Example

```
union Student {
    char name[50];
    int roll_no;
    float marks;
};
```

Union

When to use union ?

- Even though a union allows storing only one value at a time, it serves specific efficient use-cases where memory optimization is important.
- **Example**: Ideal for embedded systems or low-memory environments, When a variable can represent multiple types, but only one type at a time (like messages)

When NOT to Use Unions:

- When you need to store and access multiple fields simultaneously — use struct instead.
- When data integrity is important, and fields shouldn't overwrite each other.

Union

- Example:

```c
#include <stdio.h>

union Student {
    char name[50];
    int roll_no;
    float marks;
};

int main() {
    union Student s;

    printf("Enter student name: ");
    scanf("%s", s.name);
    printf("Name: %s\n", s.name);

    printf("Enter roll number: ");
    scanf("%d", &s.roll_no);
    printf("Roll Number: %d\n", s.roll_no);

    printf("Enter marks: ");
    scanf("%f", &s.marks);
    printf("Marks: %.2f\n", s.marks);

    printf("Name: %s\n", s.name);
    printf("Roll Number: %d\n", s.roll_no);
    printf("Marks: %.2f\n", s.marks);

    return 0;
}
```

Union Vs Structure

Structure	Union
Keyword **struct** is used.	Keyword **union** is used.

Syntax
struct structure_name
{
datatype member1;
datatype member2;
. . . . . .
datatype membern;
} ;
	Syntax:
	union union_name
	{
	datatype member1;
	datatype member2;
	. . . . . .
	datatype membern;
	};

The separate memory location is allocated to each member of the **structure.**	All members of the **union** share the same memory location.
All the members of structure can be accessed at a time.	Only one members of union can be accessed at a time.
Size of the structure is equal to the sum of size of the each member.	Size of the union is equal to the size of the largest member.

Union Vs Structure

Structure	Union

Eg.
struct employee
{
char name[20];
int age;
float salary;
};
 	
Memory reserved=(20+2+4)=26 bytes
	Eg.
	union employee
	{
	char name[20];
	int age;
	float salary;
	};
	
	Memory reserved=20 bytes

What is a Structure?

- Structure is a collection of different or similar data types variable under a common variable name. It is used for handling a group of logically related data items.
    - Examples:
        - Student name, roll number, and marks
        - Real part and complex part of a complex number
- Helps in organizing complex data in a more meaningful way
- The individual structure elements are called members.

Defining a Structure


```
struct structure_name

          {
           member 1;
           member 2;
           :
           member m;

          };
```

- struct is the required C keyword
- structure_name is the name of the structure
- member 1, member 2, … are individual member declarations.

Example:

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

Here, a derived type struct student is defined. Student is a structure name and name, roll, marks, are called structure elements or members. Each of these members belongs to different data types.

Contd.

- 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
- Once a structure has been defined, the individual structure-type variables can be declared as:

```
struct structure_name var_1, var_2, …, var_n;
```

Example

- A structure definition

```
struct student {
        char name[30];
        int  roll_number;
        int  total_marks;
        char dob[10];
             };
```

- Defining structure variables:

```
struct student  a1, a2, a3;
```

A new data-type

A Compact Form

- It is possible to combine the declaration of the structure with that of the structure variables:

```
struct tag {
        member 1;
        member 2;
        :
        member m;

        }  var_1, var_2,…, var_n;
```

- Declares three variables of type struct tag
- In this form, tag is optional

Accessing a Structure

- The members of a structure are processed individually, as separate entities
    - Each member is a separate variable.
- A structure member can be accessed by writing: variable.member

  where variable refers to the name of a structure-type variable, and member refers to the name of a member within the structure
- Examples: a1.name, a2.name, a1.roll_number, a3.dob

Accessing a Structure

- To access any member of a structure, we use the dot operator (.) .

Syntax:

`structure_variable.member`

Here structure variable refers to the name of structure type variable and member refers to name of member within structure.

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

int main()
{
  struct student st;
  strcpy(st.name,"Ram");
  st.roll=15;
  st.marks=85.5;
  printf("Name of student is %s",st.name);
  printf("\nRoll number= %d",st.roll);
  printf("\nMarks obtained=%f",st.marks);
  getch();
  return 0;
}
```

Output

Name of student is Ram
Roll number=15
Marks Obtained=85.5

Question 1

Create a structure called student having name, roll, age and address. Input the record from the user and display it.

```c
#include <stdio.h>

struct student {
    char name[50];
    int roll;
    int age;
    char address[100];
};

int main() {
    struct student s;

    printf("Enter name: ");
    gets(s.name);

    printf("Enter roll number: ");
    scanf("%d", &s.roll);

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

    printf("Enter address: ");
    gets(s.address);

    printf("Name    : %s\n", s.name);
    printf("Roll No : %d\n", s.roll);
    printf("Age     : %d\n", s.age);
    printf("Address : %s\n", s.address);

    return 0;
}
```

Question 2

Create a structure called customer having name, address, balance and accountnumber. Input a record and display its information.

```
#include <stdio.h>

struct customer {
    char name[50];
    char address[100];
    float balance;
    int accountnumber;
};

int main() {
    struct customer c;

    printf("Enter customer name: ");
    gets(c.name);

    printf("Enter address: ");
    gets(c.address);

    printf("Enter account number: ");
    scanf("%d", &c.accountnumber);

    printf("Enter account balance: ");
    scanf("%f", &c.balance);

    printf("Name           : %s\n", c.name);
    printf("Address        : %s\n", c.address);
    printf("Account Number: %d\n", c.accounnumber);
    printf("Balance        : %.2f\n", c.balance);

    return 0;
}
```

Operations on Structure Variables

- Unlike arrays, a structure variable can be directly assigned to another structure variable of the same type

```
a1 = a2;
```

  - All the individual members get assigned
- Two structure variables can not be compared for equality or inequality

```
if (a1 == a2)……
```
← this cannot be done

Array of structures

Arrays of Structures

- An array of structures is a collection of structure variables stored under one name, where each element is a structure.
- It allows you to store and manage multiple records of the same structure type (e.g., multiple students, employees, etc.).

Syntax:

```
struct StructureName {
    dataType member1;
    dataType member2;
    ...
};

struct StructureName variableName[size];
```

Arrays of Structures

- Once a structure has been defined, we can declare an array of structures

```
struct student s[50];
```

  - The individual members can be accessed as:

```
s[i].name

s[5].roll_number
```

How to declare and initialize array of structure variables?

- Suppose we want declare a structure to store 5 employee details. Array of structure to store 5 employee details is can be declared as follows:

```
struct employee
{
    char name[20];
    int id;
    float salary;
};

int main()
{
    struct employee e[5];
}
```

How to declare and initialize array of structure variables?

- Structure array initialization follows same syntax as we initialize single structure variable. The only difference is here we can initialize more than one structure variable at a time.

```
struct employee e[5] =
{
    { "Ramesh", 12,15000.35},
    { "Hari", 15, 25000.5 },
    { "Gopal", 17, 30000.25 },
    { "Sita", 2, 45000.56 },
    { "Roshan", 9, 32000.25}
};
```

Array of structures

Example:

Create a structure named student having name, roll and age. Take the input of 3 students and display the information.

```c
#include <stdio.h>

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

int main() {
    struct student s[3];
    int i;

    for (i = 0; i < 3; i++) {
        printf("Name: ");
        gets(s[i].name);
        printf("Roll: ");
        scanf("%d", &s[i].roll);
        printf("Age: ");
        scanf("%d", &s[i].age);
    }

    printf("\n--- Student Records ---\n");
    for (i = 0; i < 3; i++) {
        printf("Name: %s\n", s[i].name);
        printf("Roll: %d\n", s[i].roll);
        printf("Age: %d\n\n", s[i].age);
    }

    return 0;
}
```

Create a structure named employee having name, id and salary.
Take the input of n employees from the user and display the information.

Create a structure named employee having name, id and salary.
Take the input of n employees from the user and display the information.


```
#include <stdio.h>

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

int main()
{
int n, i;
printf("Enter the value of n");
scanf("%d", &n);

struct employee e[n];

for (i = 0; i < n; i++) {
        printf("Name: ");
        gets(s[i].name);
        printf("id: ");
        scanf("%d", &s[i].id);
        printf("Salary: ");
        scanf("%f", &s[i].salary);
    }

    for (i = 0; i < n; i++) {
        printf("Name: %s\n", s[i].name);
        printf("Id: %d\n", s[i].id);
        printf("Salary: %f\n\n", s[i].salary);
    }

    return 0;
}
```

Question:

Create a structure called customer having name, address, accountnumber and balance. Input n records and

- Display information of customer whose balance is greater than minimum balance 10000

Output:

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

struct customer {
    char name[50];
    char address[100];
    int accountNumber;
    float balance;
};

int main() {
    int n, i;

    printf("Enter number of customers: ");
    scanf("%d", &n);

    struct customer c[n];

    for (i = 0; i < n; i++) {
        printf("Name: ");
        gets(c[i].name);

        printf("Address: ");
        gets(c[i].address);

        printf("Account Number: ");
        scanf("%d", &c[i].accountNumber);

        printf("Balance: ");
        scanf("%f", &c[i].balance);
    }
```



```
printf("\n--- Customers with balance > 10000 ---\n");
    for (i = 0; i < n; i++) {
        if (c[i].balance > 10000) {
            printf("Name: %s\n", c[i].name);
            printf("Account Number: %d\n", c[i].accountNumber);
            printf("Balance: %.2f\n\n", c[i].balance);
        }
    }


    return 0;
}
```

Question:

Create a structure called customer having name, address, accountnumber and balance. Input n records and

- Display information of customer whose balance is greater than minimum balance 10000

- ask for account number from customer and display its information

Output:

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

struct customer {
    char name[50];
    char address[100];
    int accountNumber;
    float balance;
};

int main() {
    int n, i, searchAcc, found = 0;

    printf("Enter number of customers: ");
    scanf("%d", &n);

    struct customer c[n];

    for (i = 0; i < n; i++) {
        printf("Name: ");
        gets(c[i].name);

        printf("Address: ");
        gets(c[i].address);

        printf("Account Number: ");
        scanf("%d", &c[i].accountNumber);

        printf("Balance: ");
        scanf("%f", &c[i].balance);
    }
```

```
printf("\n--- Customers with balance > 10000 ---\n");
    for (i = 0; i < n; i++) {
        if (c[i].balance > 10000) {
            printf("Name: %s\n", c[i].name);
            printf("Account Number: %d\n", c[i].accountNumber);
            printf("Balance: %.2f\n\n", c[i].balance);
        }
    }

    printf("Enter account number to search: ");
    scanf("%d", &searchAcc);

    for (i = 0; i < n; i++) {
        if (c[i].accountNumber == searchAcc) {
            printf("\n--- Customer Found ---\n");
            printf("Name: %s\n", c[i].name);
            printf("Address: %s\n", c[i].address);
            printf("Account Number: %d\n", c[i].accountNumber);
            printf("Balance: %.2f\n", c[i].balance);
            found = 1;
            break;
        }
    }

    if (!found) {
        printf("No customer found with account number %d\n", searchAcc);
    }

    return 0;
}
```

Arrays within Structures

- A structure member can be an array

```
struct  student
{
      char  name[30];
      int  roll_number;
      int  marks[5];
      char  dob[10];
}  a1, a2, a3;
```

- The array element within the structure can be accessed as:

```
a1.marks[2], a1.dob[3],…
```

Arrays within Structures


```
#include <stdio.h>

struct Student {
    char name[50];
    int marks[5];
};

int main() {
    struct Student s1 = {"John", {85, 90, 78, 92, 88}};

    printf("Name: %s\n", s1.name);
    printf("Marks: ");
    for(int i = 0; i < 5; i++) {
        printf("%d ", s1.marks[i]);
    }
    printf("\n");

    return 0;
}
```

Array of structures

Create a structure called employee having name, address, salary and age. Input n records of employee and display information of employee whose address is Kathmandu.

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
struct employee
{
char name[20];
char address[20];
float salary;
int age;
};

int main()
{
  struct employee e[100];
  int i,n;
  printf("Enter how many records you want to enter");
  scanf("%d",&n);
  for(i=0;i<n;i++)
  {
    printf("Enter the records of employee %d\n",i+1);
    printf("Enter the name");
    gets(e[i].name);
    printf("Enter the address");
    gets(e[i].address);
    printf("Enter the salary");
    scanf("%f",&e[i].salary);
    printf("Enter the age");
    scanf("%d",&e[i].age);
}
```

Array of structures

Create a structure called employee having name, address, salary and age. Input n records of employee and display information of employee whose address is Kathmandu.

```
printf("\nInformation of employee whose address is kathmandu
are\n");

printf("\nName\tAddress\tSalary\tAge");

for(i=0;i<n;i++)
{
  if(strcmp(e[i].address,"kathmandu")==0)
  {
   printf("\n%s\t%s\t%f\t%d",e[i].name,e[i].address,
   e[i].salary,e[i].age);
  }
}
getch();
return 0;
}
```

Create a structure called book, name, price, author, and published date in day, month, and year. Write a program to read 100 books information from the user and displays those records having price greater than 250.

Solution:


```
#include<stdio.h>
#include<conio.h>
struct date
{
  int dd;
  int mm;
  int yy;
};

struct book
{
  char name[20];
  float price;
  char author[20];
  struct date d;
};
```

```
int main()
{
int i;
struct book b[100];
for(i=0;i<100;i++)
{
printf("Enter the information of book %d\n",i+1);
printf("Enter the book name");
gets(b[i].name);
printf("Enter the author name");
gets(b[i].author);
printf("Enter the book price");
scanf("%f",&b[i].price);
printf("Enter book published year");
scanf("%d",&b[i].d.yy);
printf("Enter book published month");
scanf("%d",&b[i].d.mm);
printf("Enter book published date");
scanf("%d",&b[i].d.dd);
}
```

Solution:

```
printf("\nInformation of Books having price greater than 250 are\n");
printf("\nName\tPrice\tAuthor\tPublished date");

for(i=0;i<100;i++)
{
    if(b[i].price>250)
    {
printf("\n%s\t%f\t%s\t%d\t%d\t%d",b[i].name,b[i].price,b[i].author,b[i
].d.dd,b[i].d.mm, b[i].d.yy);
     }
}
getch();
return 0;
}
```

Array of Structures Vs Structure Containing Arrays.

Array of Structures Vs Structure Containing Arrays.

Array of Structures

- It is an array where each element is a structure.
- Used to store multiple records (each record is a structure).
- Example: Storing data of multiple students, employees, etc.

Structure Containing Arrays

- It is a structure that contains arrays as its members.
- Used when one record contains multiple related data items (stored in arrays).
- Example: Storing marks of a student in multiple subjects, storing multiple phone numbers, etc.

Example: Array of 3 structures

```
#include <stdio.h>
struct Student {
    int roll;
    char name[50];
    float marks;
};

int main() {
    struct Student s[3];
    for (int i = 0; i < 3; i++) {
        printf("Enter roll, name and marks for student %d:\n", i+1);
        scanf("%d %s %f", &s[i].roll, s[i].name, &s[i].marks);
    }

    printf("\nStudent Details:\n");
    for (int i = 0; i < 3; i++) {
        printf("Roll: %d, Name: %s, Marks: %.2f\n", s[i].roll, s[i].name,
s[i].marks);
    }

    return 0;
}
```

Example: Structure Containing Arrays

```c
#include <stdio.h>
struct Student {
    int roll;
    char name[50];
    float marks[5]; // Array of marks for 5 subjects
};

int main() {
    struct Student s1;

    printf("Enter roll number and name:\n");
    scanf("%d %s", &s1.roll, s1.name);
    printf("Enter marks for 5 subjects:\n");
    for (int i = 0; i < 5; i++) {
        scanf("%f", &s1.marks[i]);
    }

    printf("\nStudent Details:\n");
    printf("Roll: %d, Name: %s\n", s1.roll, s1.name);
    printf("Marks:\n");
    for (int i = 0; i < 5; i++) {
        printf("%.2f ", s1.marks[i]);
    }
    return 0;
}
```

Nested Structure

Nested Structure

- A **nested structure** means a structure **within another structure**.
- It allows grouping related data more logically and hierarchically.

Syntax:

```
struct Inner {
    // members
};

struct Outer {
    // other members
    struct Inner innerVar;  // Nested structure
};
```

Nested Structure

- Example:

```
#include<stdio.h>
#include<conio.h>

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

struct employee
{
char name[20];
int id;
struct date dob;
float salary;
}e;
```

Nested Structure

- Example:

```
int main()
{
printf("Enter the name of employee");
scanf("%s",e.name);

printf("Enter ID of employee");
scanf("%d",&e.id);

printf("Enter the year of birthday\n");
scanf("%d",&e.dob.year);

printf("Enter the month of birthday\n");
scanf("%d",&e.dob.month);

printf("Enter the day of birthday\n");
scanf("%d",&e.dob.day);

printf("Enter the salary of employee\n");
scanf("%f",&e.salary);

printf("Name\tId\tDay\tMonth\tYear\tsalary\n");
printf("%s\t%d\t%d\t%d\t%d\t%f",e.name,e.id,e.dob.day,e
.dob.month,e.dob.year,e.salary);
getch();
return 0;
}
```

Pointer to Structure

Pointer to Structure

- Pointers can also be used pointing to a structure.
- We can define pointers to structures in the following way

```
struct structure_name *pointer_name;
```

- We can store the address of a structure variable in following way:

```
pointer_name = &structure_variable name;
```

- To access the members of a structure using a pointer to that structure, you must use the → operator as follows:

```
pointer_name->member;
```

Pointer to Structure

- Example:

```
#include<stdio.h>
#include<conio.h>
struct book
{
char name[30];
int pages;
float price;
};

int main()
{
struct book b={"C programming",300,550.30};
struct book *ptr;
ptr=&b;
printf("\nBook name\tNo.of pages\tprice\n");
printf("\n%s\t%d\t%f",ptr->name,ptr->pages,ptr->price);
getch();
return 0;
}
```

Structure and Function

In C , structure can be passed to functions by two methods.

1) Passing by value (Passing actual value as argument)

2) Passing by reference (Passing address of an argument)

Parameter Passing in a Function

- Structure variables can be passed as parameters like any other variables. Only the values will be copied during function invocation.

```
void swap (struct complex a, struct complex b)
{
    struct complex tmp;

    tmp=a;
    a=b;
    b=tmp;
}
```

Structure and Function : Passing by value

- This methods involves passing a copy of the entire structure to the called function. Any changes to structure members within the function are not reflected in original structure in the calling function.

```
#include<stdio.h>
#include<conio.h>
struct employee
{
char name[50];
float salary;
};
void display(struct employee e);
void addbonus(struct employee ee);

int main()
{
struct employee emp;
printf("Enter the name of employee");
scanf("%s",emp.name);
printf("Enter the salary of employee");
scanf("%f",&emp.salary);
addbonus(emp);
display(emp);
getch();
}
```

Structure and Function : Passing by value

- This methods involves passing a copy of the entire structure to the called function. Any changes to structure members within the function are not reflected in original structure in the calling function.

```
.void addbonus(struct employee ee)
 {
 ee.salary=ee.salary+5000;
 }

 void display(struct employee e)
 {
 printf("\nInformation of Employee");
 printf("\nName:%s",e.name);
 printf("\nSalary:%f",e.salary);
 }
```

Output

Enter the name of employee: Ramesh
Enter the salary of employee: 25000
Information of employee
Name: Ramesh
Salary: 25000

Structure and Function : Passing by reference

- In this method the address location of the structure is passed to the called function. The function can access indirectly the entire structure and work on it. Any changes made in structure variable in function definition reflects in original structure variable in calling function.

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

void display(struct employee *e);
void addbonus(struct employee *ee);

int main()
{
struct employee emp;
printf("Enter the name of employee");
scanf("%s",emp.name);
printf("Enter the salary of employee");
scanf("%f",&emp.salary);
addbonus(&emp);
display(&emp);
getch();
}
```

Structure and Function : Passing by reference

- In this method the address location of the structure is passed to the called function. The function can access indirectly the entire structure and work on it. Any changes made in structure variable in function definition reflects in original structure variable in calling function.

```
void addbonus(struct employee *ee)
{
ee->salary=ee->salary+5000;
}

void display(struct employee *e)
{
printf("\nInformation of Employee");
printf("\nName:%s",e->name);
printf("\nSalary:%f",e->salary);
}
```

Output:

Enter the name of employee: Ramesh
Enter the salary of employee: 25000
Information of employee
Name: Ramesh
Salary: 30000

Returning structures

- It is also possible to return structure values from a function. The return data type of the function should be as same as the data type of the structure itself

```
struct complex add(struct complex a, struct complex b)
{
    struct complex tmp;

    tmp.real = a.real + b.real;
    tmp.imag = a.imag + b.imag;
    return(tmp);
}
```

Direct arithmetic operations are not possible with structure variables

Write a program in C to add two complex numbers using structure and function.


```
#include <stdio.h>

struct complex {
    float real;
    float imag;
};

struct complex add(struct complex a, struct complex b) {
    struct complex tmp;
    tmp.real = a.real + b.real;
    tmp.imag = a.imag + b.imag;
    return tmp;
}
```

Write a program in C to add two complex numbers using structure and function.


```
int main() {
    struct complex num1, num2, result;

    printf("Enter real and imaginary part of first complex number:\n");
    printf("Real: ");
    scanf("%f", &num1.real);
    printf("Imaginary: ");
    scanf("%f", &num1.imag);

    printf("\nEnter real and imaginary part of second complex number:\n");
    printf("Real: ");
    scanf("%f", &num2.real);
    printf("Imaginary: ");
    scanf("%f", &num2.imag);

    result = add(num1, num2);

    printf("\nSum of complex numbers = %.2f + %.2fi\n", result.real, result.imag);

    return 0;
}
```

Write a program in C to add two complex numbers using structure and function.

Output:

```
Enter real and imaginary part of first complex number:
Real: 3.2
Imaginary: 4.5

Enter real and imaginary part of second complex number:
Real: 1.8
Imaginary: 2.3

Sum of complex numbers = 5.00 + 6.80i
```

Union

Union

- A union is a user-defined data type in C that allows storing different data types in the same memory location. However, only one member can contain a value at any given time—all members share the same memory space.

Defining a Union

```
union union_name
                    {
        member 1;
        member 2;
        :
        member m;
       };
```

union is the C keyword
union_name is the name of the structure
member 1, member 2, … are individual member declarations.

Example

```
union Student {
    char name[50];
    int roll_no;
    float marks;
};
```

Union

When to use union ?

- Even though a union allows storing only one value at a time, it serves specific efficient use-cases where memory optimization is important.
- **Example**: Ideal for embedded systems or low-memory environments, When a variable can represent multiple types, but only one type at a time (like messages)

When NOT to Use Unions:

- When you need to store and access multiple fields simultaneously — use struct instead.
- When data integrity is important, and fields shouldn't overwrite each other.

Union

- Example:

```c
#include <stdio.h>

union Student {
    char name[50];
    int roll_no;
    float marks;
};

int main() {
    union Student s;

    printf("Enter student name: ");
    scanf("%s", s.name);
    printf("Name: %s\n", s.name);

    printf("Enter roll number: ");
    scanf("%d", &s.roll_no);
    printf("Roll Number: %d\n", s.roll_no);

    printf("Enter marks: ");
    scanf("%f", &s.marks);
    printf("Marks: %.2f\n", s.marks);

    printf("Name: %s\n", s.name);
    printf("Roll Number: %d\n", s.roll_no);
    printf("Marks: %.2f\n", s.marks);

    return 0;
}
```

Union Vs Structure

Structure	Union
Keyword **struct** is used.	Keyword **union** is used.

Syntax
struct structure_name
{
datatype member1;
datatype member2;
. . . . . .
datatype membern;
} ;
	Syntax:
	union union_name
	{
	datatype member1;
	datatype member2;
	. . . . . .
	datatype membern;
	};

The separate memory location is allocated to each member of the **structure.**	All members of the **union** share the same memory location.
All the members of structure can be accessed at a time.	Only one members of union can be accessed at a time.
Size of the structure is equal to the sum of size of the each member.	Size of the union is equal to the size of the largest member.

Union Vs Structure

Structure	Union

Eg.
struct employee
{
char name[20];
int age;
float salary;
};
 	
Memory reserved=(20+2+4)=26 bytes
	Eg.
	union employee
	{
	char name[20];
	int age;
	float salary;
	};
	
	Memory reserved=20 bytes

Questions

1. Define a structure in C. Write a simple program to declare a structure for a student record with roll number, name, and marks.
2. Explain how structure elements are accessed in C with an example.
3. How can you initialize a structure during its declaration? Explain with an example.
4. Differentiate between array of structures and structure containing arrays with examples.
5. What is a nested structure? Write a small code snippet demonstrating nested structures.
6. Explain how structures can be passed to functions in C. Give an example using call by value.
7. What is a union in C? How is it different from a structure?
8. Differences between structures and unions.

THANK YOU

Any Queries ?