Unit 9

Structure and Union

Introduction

- Structure is a collection of variables (can be of different types) under a single name.
- **For example:** You want to store information about a person: his/her name, citizenship number and salary. You can create different variables name, citNo and salary to store these information separately.
- What if you need to store information of more than one person? Now, you need to create different variables for each information per
  person: name1, citNo1, salary1, name2, citNo2, salary2 etc.
- A better approach would be to have a collection of all related information under a single name Person structure, and use it for every person.

How to define a structure?

- Keyword struct is used for creating a structure.
- Syntax

```
struct structure_name
{
   data_type member1;
   data_type member2;
   .
   .
   data_type memeber;
};
```

Example

```
struct Person
{
    char name[50];
    int citNo;
    float salary;
};
```

Create structure variable

- When a structure is defined, it creates a user-defined type. However, no storage or memory is allocated. To allocate memory of a given structure type and work with it, we need to create variables.

```
struct Person
{
    char name[50];
    int citNo;
    float salary;
}
void main()
{
    struct Person person1, person2, p[20];
}
```

- Another way of creating a structure variable is

```
struct Person
{
    char name[50];
    int citNo;
    float salary;
} person1, person2, p[20];
```

How to Access members of a structure?

- There are two types of operators used for accessing members of a structure.
- Member operator(.)
- Structure pointer operator(->) (will be discussed in structure and pointers)
- Suppose, you want to access salary of person2. Here's how you can do it:
- person2.salary

Program to add two distances which is in feet and inches

```
#include <stdio.h>
struct Distance
{
    int feet;
    float inch;
} dist1, dist2, sum;
void main()
{
    printf("1st distance\n");

    printf("Enter feet: ");
    scanf("%d", &dist1.feet);
```

```
printf("Enter inch: ");
scanf("%f", &dist1.inch);

printf("2nd distance\n");

printf("Enter feet: ");
scanf("%d", &dist2.feet);

printf("Enter inch: ");
scanf("%f", &dist2.inch);
```

```
 // adding feet
    sum.feet = dist1.feet + dist2.feet;
    // adding inches
    sum.inch = dist1.inch + dist2.inch;
    // changing feet if inch is greater than 12
    while (sum.inch >= 12)
    {
        sum.feet++;
        sum.inch = sum.inch - 12;
    }
    printf("Sum of distances = %d\'-%.1f\"", sum.feet, sum.inch);
}
```

Structure array

```
#include <stdio.h>
struct student
{
   int roll;
   char name[50];
   float marks;
} s[10];
```

```
int main()
{
   int i;
   printf("Enter information of 10 students:\n");
   for(i=0; i<10; ++i)
   {
      printf("\nenter roll no");
      scanf("%d",&s[i].roll);
      printf("\nEnter name: ");
      gets(s[i].name);
      printf("\nEnter marks: ");
      scanf("%f",&s[i].marks);
      printf("\n");
   }
```

```
printf("Displaying Information:\n\n");
    // displaying information
    for(i=0; i<10; ++i)
    {
        printf("\nRoll number: %d\n",s[i].roll);
        printf("\nName: is %s ",s[i].name);
        printf("\nMarks: %.2f",s[i].marks);
        printf("\n");
    }
    return 0;
```

Write a program to store the records of N customer in a bank with fields account_no, name and balance. Read the records of N customers from user and display the records of the customer who has the highest balance in the bank.

```
#include <stdio.h>
struct customer
{
   int acc_no;
   char name[50];
   float balance;
} ;
void main()
{
   int i,n,index;
   float max_balance=0;
   struct customer c[100];

   printf("how many customers");
   scanf("%d",&n);
```

```
for(i=0; i<n; ++i)
  {
     printf("\nenter data of %d customer",i+1);
     printf("\nenter account no");
     scanf("%d",&c[i].acc_no);

     printf("\nEnter name: ");
     fflush(stdin);
     gets(c[i].name);

     printf("\nEnter balance: ");
     scanf("%f",&c[i].balance);
     printf("\n");
  }
  max_balance=c[0].balance;
```

```
 index=0;
   for(i=0; i<n; i++)
   {
     if(c[i].balance>max_balance)
     {
          max_balance=c[i].balance;
          index=i;
          }
     }
   printf("Customer having the highest balance is \n");
   printf("\naccount number: %d\n",c[index].acc_no);
   printf("\nName: is %s ",c[index].name);
   printf("\nBalance: %.2f",c[index].balance);
   printf("\n");
}
```

- Write a program to store the records of N students with fields rollno, name and address. Read the records of N students from user and display the records of the students who are from kathmandu.

```
#include <stdio.h>
struct student
{
    int roll;
    char name[50];
    char address[50];
} s[100];
int main()
{
    int i,n;
    printf("Enter how many students??\n");
    scanf("%d",&n);
```

```
for(i=0; i<n; ++i)
  {
     printf("\nenter roll no");
     scanf("%d",&s[i].roll);

     printf("\nEnter name: ");
     fflush(stdin);
     gets(s[i].name);

     printf("\nEnter address: ");
     fflush(stdin);
     gets(s[i].address);
  }
```

```
printf("Displaying Information of person from kathmandu: \n\n");
   printf("\n");
   for(i=0; i<n; ++i)
   {
     if(strcmp(s[i].address,"kathmandu")==0)
     {
           printf("\nRoll number: %d\n",s[i].roll);
           printf("\nName: is %s ",s[i].name);
           printf("\naddress: %.2f",s[i].address);
           printf("\n");
          }
   }
   return 0;
}
```

Structure and Function

- **Passing structure member to function**

```
#include <stdio.h>
struct student
{
   int roll;
   char name[50];
} s;
void display(int ,char [20]);
int main()
{
   printf("\nenter roll no");
   scanf("%d",&s.roll);
   printf("\nEnter name: ");
   fflush(stdin);
   gets(s.name);
   display(s.roll,s.name);
}
void display(int roll,char name[20])
{
     printf("\nRoll number: %d\n",roll);
     printf("\nName: is %s ",name);
}
```

C Structure and Function

- **passing structure as an argument**

```
#include <stdio.h>
struct student
{
   char name[50];
   int age;
};
// function prototype
void display(struct student s);
```

```
int main()
{
    struct student s1;
    printf("Enter name:");
    gets(s1.name);
    printf("Enter age:");
    scanf("%d", &s1.age);

    display(s1);   // passing structure as an argument
    return 0;
}
```

```
void display(struct student s)
{
  printf("\nDisplaying information\n");
  printf("Name: %s", s.name);
  printf("\nRoll: %d", s.age);
}
```

Returning structure from a function

```
#include <stdio.h>
struct student
{
   char name[50];
   int age;
};
// function prototype
struct student getInformation();
```

```
int main()
{
    struct student s;
    s = getInformation();
    printf("\nDisplaying information\n");
    printf("Name: %s", s.name);
    printf("\nRoll: %d", s.age);

    return 0;
}
```

```
struct student getInformation()
{
  struct student s1;
  printf("Enter name:");
  gets(s1.name);
  printf("Enter age:");
  scanf("%d", &s1.age);

  return s1;
}
```

C Programming Structure and Pointer

```
struct name {
   member1;
   member2;
   .
   .
};

int main()
{
   struct name *ptr, Harry;
}
```

Example

```
#include <stdio.h>
struct person
{
   int age;
   float weight;
};
int main()
{
    struct person *personPtr, person1;
    personPtr = &person1;

    printf("Enter age:");
    scanf("%d", &personPtr->age);
    printf("Enter weight:");
    scanf("%f", &personPtr->weight);

    printf("Displaying:\n");
    printf("Age: %d\n", personPtr->age);
    printf("weight: %f", personPtr->weight);
    return 0;
}
```

- In this example, the address of person1 is stored in personPtr pointer variable using code personPtr = &person1;.
- Now, you can access members of person1 using personPtr pointer. For that we use -> operator.
- By the way,
- personPtr->age is equivalent to (*personPtr).age
- personPtr->weight is equivalent to (*personPtr).weight

Passing structure reference

```
#include<stdio.h>
struct student
{
     int roll;
     char name[20];
};
void display(struct student *);
int main()
{
   struct student s1;
   printf("\nenter roll");
```

```
 scanf("%d",&s1.roll);
   printf("\nenter name");
   fflush(stdin);
   gets(s1.name);
   display(&s1);
}
void display(struct student *s)
{
      printf("\nid is %d",s->roll);
      printf("\nname is %s",s->name);
}
```

Nested Structure

- C provides us the feature of nesting one structure within another structure by using which, complex data types are created.
- For example, we may need to store the address of an entity employee in a structure. The attribute address may also have the subparts as street number, city, state, and pin code.
- Hence, to store the address of the employee, we need to store the address of the employee into a separate structure and nest the structure address into the structure employee.
- The structure can be nested in the following ways.
  - By separate structure
  - By Embedded structure

Example of separate structure

```
#include<stdio.h>
struct address
{
   char city[20];
   int pin;
   char phone[14];
};
struct employee
{
   char name[20];
   struct address add;
};
```

```
void main ()
{
    struct employee emp;
    printf("Enter employee information?\n");
    printf("\nenter name");
    gets(emp.name);
    printf("\nenter city");
    gets(emp.add.city);
    printf("\nenter pincode");
    scanf("%d",&emp.add.pin);
    printf("\nenter phone");
    fflush(stdin);
    gets(emp.add.phone);
    printf("\nPrinting the employee information....\n");
    printf("name: %s\nCity: %s\nPincode: %d\nPhone: %s",
emp.name,emp.add.city,emp.add.pin,emp.add.phone);
}
```

Embedded Nested Structure

```
struct employee
{
    char name[20];
    struct address
    {
        char city[20];
        int pin;
        char phone[14];
    }add;
};
```

Union

- Union is a user-defined type similar to a structure in C programming
- **How to define a union?**

```
union car
{
  char name[50];
  int price;
};
```

Create union variables

- When a union is defined, it creates a user-defined type. However, no memory is allocated. To allocate memory for a given union type and work with it, we need to create variables.

```
union car
{
  char name[50];
  int price;
};
int main()
{
  union car car1, car2, *car3;
  return 0;
}
```

Another way

```
union car
{
  char name[50];
  int price;
} car1, car2, *car3;
```

How to access members of a union?

- We use . to access normal variables of a union. To access pointer variables, we use -> operator.
- In the above example,
- price for car1 can be accessed using car1.price
- price for car3 can be accessed using car3->price

Example

```
#include <stdio.h>
union unionJob
{
  //defining a union
  char name[32];
  float salary;
  int workerNo;
} uJob;
struct structJob
{
  char name[32];
  float salary;
  int workerNo;
} sJob;
```

```
int main()
{
   printf("size of union = %d bytes", sizeof(uJob));
   printf("\nsize of structure = %d bytes", sizeof(sJob));
   return 0;
}
```

- **Why this difference in size of union and structure variables?**
- The size of structure variable is 40 bytes. It's because:
- size of name[32] is 32 bytes
- size of salary is 4 bytes
- size of workerNo is 4 bytes
- However, the size of union variable is 32 bytes. It's because the size of union variable will always be the size of its largest element. In the above example, the size of largest element (name[32]) is 32 byes.

- **Only one union member can be accessed at a time**
- You can access all members of a structure at once as sufficient memory is allocated for all members. However, it's not the case in unions. Let's see an example:

```
#include <stdio.h>
union Job
{
   float salary;
   int workerNo;
} j;
```

```
int main()
{
   j.salary = 12.3;
   j.workerNo = 100;
   printf("Salary = %.1f\n", j.salary);
   printf("Number of workers = %d", j.workerNo);
   return 0;
}
```

Output:

```
Salary = 0.0
Number of workers = 100
```