Structure

Unit 10

Introduction

A structure is a collection of logically related data items grouped together under a single name, called structure tag.

The data items enclosed within a structure are known as members. The members can be of same or different data types

```
struct structure_name

{

        data_type member1;

        data_type member2;

        data_type member3;

}
```

Example of Structure

```
struct employee {
        int emp_id;
        char name[25];
        int age;
        float salary;
} e1, e2;
```

The members of a structure do not occupy member until they are associated with a structure variable. Above examples shows the structure variable e1 and e2 of a structure employee.

Create struct variables

When a `struct` type is declared, no storage or memory is allocated. To allocate memory of a given structure type and work with it, we need to create variables.

Here's how we create structure variables:

```
struct Person {

  // code

};

int main() {

  struct Person person1, person2, p[20];

  return 0;

  }
```

Access member of structure

There are two types of operators used for accessing members of a structure.

1.	`.`	Member operator
2.	`->`	Structure pointer operator

```
#include <stdio.h>
#include <string.h>
// create struct with person1 variable
struct Person {
  char name[50];
  int citNo;
  float salary;
} person1;

int main() {
  // assign value to name of person1
  strcpy(person1.name, "George Orwell");
  // assign values to other person1 variables
  person1.citNo = 1984;
  person1. salary = 2500;

  // print struct variables
  printf("Name: %s\n", person1.name);
  printf("Citizenship No.: %d\n", person1.citNo);
  printf("Salary: %.2f", person1.salary);

  return 0;
}
```

```
#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;
}
```

Array of Structure

```
#include <stdio.h>
#include<string.h>
struct person
{
    char name[20];
   int age;
   float weight;
};

int main()
{
    int i;
    struct person p[50];
    for(i=0;i<3;i++)
    {
        printf("Enter name, age and weight:");
        scanf("%s %d %f",p[i].name,&p[i].age,&p[i].weight);

    }
 printf("Data of the person:\n");
   for(i=0;i<3;i++)
   {
       printf("Person %d\n",i+1);
       printf("Name: %s\n",p[i].name);
       printf("Age: %d\n",p[i].age);
       printf("Weight: %.2f\n",p[i].weight);
   }
   return 0;
}
```

Structure and Function

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

int main() {
   struct student s1;

   printf("Enter name: ");
   scanf("%[^\n]s", s1.name);

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

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

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

Pointer function and Structure

```
#include <stdio.h>
typedef struct Complex
{
   float real;
   float imag;
}complex;

void addNumbers(complex c1, complex c2, complex *result);

int main()
{
   complex c1, c2, result;

   printf("For first number,\n");
   printf("Enter real part: ");
   scanf("%f", &c1.real);
   printf("Enter imaginary part: ");
   scanf("%f", &c1.imag);

 printf("For second number, \n");
   printf("Enter real part: ");
   scanf("%f", &c2.real);
   printf("Enter imaginary part: ");
   scanf("%f", &c2.imag);

   addNumbers(c1, c2, &result);
   printf("\nresult.real = %.1f\n", result.real);
   printf("result.imag = %.1f", result.imag);

   return 0;
}
void addNumbers(complex c1, complex c2,
complex *result)
{
    result->real = c1.real + c2.real;
    result->imag = c1.imag + c2.imag;
}
```

Nested Structure

Structure within the structures are known as nested structures. You can define the nested structure in any of the two below formats.

```
struct Organisation
{
  char organisation_name[20];
  char org_number[20];

  struct Employee
  {
    int employee_id;
    char name[20];
    int salary;
  }emp;

};
```

```
struct Employee
  {
    int employee_id;
    char name[20];
    int salary;
  };
struct Organisation
{
  char organisation_name[20];
  char org_number[20];
    struct Employee emp;
};
```

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

struct Employee
{
	int employee_id;
	char name[20];
	int salary;
};

struct Organisation
{
	char organisation_name[20];
	char org_number[20];

	struct Employee emp;
};
```

```
int main()
{
struct Organisation org;

printf("The size of structure organisation :
%ld\n",sizeof(org));

org.emp.employee_id = 101;
strcpy(org.emp.name, "Francis");
org.emp.salary = 400000;
strcpy(org.organisation_name,
		"Google");
strcpy(org.org_number, "G12345");

// Printing the details
printf("Organisation Name : %s\n",org.organisation_name);
printf("Organisation Number : %s\n",org.org_number);
printf("Employee id : %d\n",org.emp.employee_id);
printf("Employee name : %s\n",org.emp.name);
printf("Employee Salary : %d\n",org.emp.salary);
}
```

Unions

A union is a user-defined type similar to structs in C except for one key difference.

Structures allocate enough space to store all their members, whereas unions can only hold one member value at a time.

```
union car{

  char name[50];

  int price;

};

int main()

{

  union car car1, car2, *car3;

  return 0;

  }
```

Structure	Union
Keyword **struct** defines structure	keyword **union** defines Union
`struct mystructure{ int id; char name[20]; float age; }s1;`	`struct myunion{ int id; char name[20]; float age; }u1;`
Within a structure all member gets allocated and members have address that increase as the declarators are read left to right.The total size of the structure is the sum of all the members.	For a union compiler allocates the memory for the largest of all member.
Within a structure all members gets memory allocated; therefore any member can be retrieved at any time.	While retrieving data from a union the type that is being retrieved must be the type most recently stored.
One or more members of structure can be initialised at once.	A union may only be initialised with a value of the type of its first member.

```
#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;
   }
```

```
#include <stdio.h>

union Job {

   float salary;

   int workerNo;

} j;

int main() {

   j.salary = 12.3;

   // when j.workerNo is assigned a value,

   // j.salary will no longer hold 12.3

   j.workerNo = 100;

   printf("Salary = %.1f\n", j.salary);

   printf("Number of workers = %d", j.workerNo);

   return 0;

  }
```

Create a structure and display the details of fees less than 1000

```
#include <stdio.h>
struct students{
        char name[30];
        int age;
        float fees;
};
int main()
{
        struct students s[10];
        int i,a;
        float max;
        for(i=0;i<5;i++)  {
        printf("Enter student %d details\n",i+1);
        printf("Enter name and age:");
        scanf("%s %d %f",s[i].name,&s[i].age,&s[i].fees);

        }
        printf("Enter a price:\n");
        scanf("%f",&max);
printf("Displaying the result\n");
        for(i=0;i<5;i++)
        {
        if(s[i].fees<=max){
        printf("student %d details:\n",i+1);
        printf("Name:%s\n",s[i].name);
        printf("Age:%d\n",s[i].age);
        }

        }


        return 0;
}
```

Create a structure named employee and display the result based on the salary(sorted ascending order)

```
#include <stdio.h>
struct Employee{
        char name[50];
        int id;
        int age;
        float salary;
}e[100],e1[100];
int main()
{
        float temp;
        int index;
        for(int i=0;i<5;i++)
        {
        printf("Enter the information of Employee %d\n",i+1);
        printf("Enter the name, id, age and salary of the employee:\n");
        scanf("%s %d %d %f",e[i].name,&e[i].id,&e[i].age,&e[i].salary);
        }
```

```
for(int i=0;i<5;i++)
        {
        e1[i]=e[i];
        }
for(int i=0;i<5;i++)
        {
        for(int j=i+1;j<5;j++)
        {
        if(e[i].salary>e[j].salary)
        {
                temp=e[i].salary;
                e[i].salary = e[j].salary;
                e[j].salary = temp;
        }
        }
        }
```

Displaying the result

```
for(int i=0;i<5;i++)
        {
        for(int j=0;j<5;j++)
        {
        if(e[i].salary==e1[j].salary)
                index=j;
        }

   //   printf("Displaying the information of Employee %d\n",i+1);
        printf("Name:%s\n id:%d\n age:%d\n  salary:%.2f\n",e1[index].name ,e1[index].id
,e1[index].age,e1[index].salary);

        }
        return 0;
}
```

Self Referential Structure

- A self-referential structure is a structure that can have members which point to a structure variable of the same type.
- They can have one or more pointers pointing to the same type of structure as their member.
- The self-referential structure is widely used in dynamic data structures such as trees, linked lists, and so on.
- The next node of a node will be pointed in linked lists, which consists of the same struct type.
- It is a unique type of structure containing a member of its type. The member of its type is a pointer variable of the same structure in which it has been declared.
- In the context of blockchain, each block is linked to a previous node or a next node, similar to a linked list.

Syntax:

```
struct structure_name

 {

 datatype datatype_name;

 structure_name * pointer_name;

  }
```

E.g.:

```
struct node

 {

 int data;

 struct node *next;

  };
```

Where ‘next’ is a pointer to a struct node variable, it should be remembered that the pointer to a structure is similar to the pointer to any other variable. Hence, self-referential data structure is generally a structure definition that includes at least one member that is a pointer to the structure of its kind.