Shastri 5th Semester

Computer Science

Unit: 4th

C LANGUAGE: STRUCTURES AND UNIONS

In C programming, a structure (also known as a struct) is a user-defined data type that can store a collection of variables of different data types. These variables are called members, and they are accessed using the dot operator (.). For example, if you have a struct called "Person" that has members for a name, age, and address, you can access the name of a Person variable by using the dot operator: variableName.name.

A union, on the other hand, is also a user-defined data type, but it differs from a struct in that all its members share the same memory location. This means that the size of a union is the size of its largest member, and when you access one member, you may be overwriting the values of the other members. Unions are typically used to save memory when you need to store different data types in the same location, but only one at a time.

Structures and unions are both used to store multiple data elements of different types, but they store these data elements in different ways. Structures store the data elements in separate memory locations, while unions store them in the same memory location.

Structures	**Unions**
A structure is a user-defined data type that groups together different data types into a single unit.	A union is a user-defined data type that stores different data types in the same memory location.
Each member of a structure has its own memory space.	All members of a union share the same memory space.
The size of a structure is the sum of the sizes of all its members.	The size of a union is the size of its largest member.

A structure can have different members with different data types.	A union can only have one member active at a time, as all members share the same memory space.

Structure variables, initialization, structure assignment

Structure variables:

A structure variable is a variable that can hold all the members of a structure.

Initialization:

Structures can be initialized when they are defined or when they are declared.

Example:

```
struct point {
   int x;
   int y;
};
struct point p1 = {1, 2}; // Initialization when defined
struct point p2 = {3, 4}; // Initialization when defined
struct point p3; // Declaration
p3 = {5, 6}; // Initialization when declared
```

Structure assignment:

You can assign the values of one structure variable to another structure variable of the same type using the assignment operator (=).

Example:

```
struct point p1 = {1, 2};
struct point p2;
p2 = p1; // Assigning the values of p1 to p2
```

Note: when you assign one structure to another, the entire structure is copied, not just the pointer to the structure.

C Language: Nested Structure

A nested structure is a structure that is defined inside another structure. It is also known as a "member structure."

Example

```
struct outer {
    int a;
    struct inner {
        int b;
        int c;
    } i;
};
```

In the above example, the structure "inner" is nested within the structure "outer."

To access the members of the nested structure, you need to use the dot operator twice, once to access the nested structure, and then to access the member of the nested structure.

Example:

Example

```
struct outer o;
o.i.b = 5;
o.i.c = 10;
```

Nested structures are useful when you want to group related data together, for example, you can use nested structures to represent a date, where the outer structure contains the day, month and year and inner structure can be used to represent the time with hours, minutes and seconds.

Example

```
struct date {
   int day;
   int month;
   int year;
   struct time {
      int hours;
      int minutes;
      int seconds;
   } t;
};
```

It's also useful when you have a large number of related data and you want to organize them in a logical way.

Functions Structures and Arrays

Functions and structures are both user-defined data types in C and C++ programming languages, while arrays are built-in data types.

Functions:

A function is a block of code that performs a specific task. Functions can take input parameters and return a value. Functions are used to organize code into reusable and modular chunks, which improves readability and maintainability of the code.

Example:

```
int add(int a, int b) {
   return a + b;
}
```

Structures:

A structure is a user-defined data type that groups together different data types into a single unit. Structures are used to represent complex data types, such as a point in 2D space, a date, or a person's information.

Example:

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

Arrays:

An array is a built-in data type that stores a collection of items of the same type. Arrays are used to store and manipulate large amounts of data. The items in an array are accessed by their index, which is a zero-based integer.

Example:

```
int numbers[10];
```

Functions can take structures and arrays as input parameters, and they can also return structures and arrays as output. Structures can also contain arrays as members.

Functions can be used to manipulate arrays, for example, you can create a function that sorts an array, a function that searches for an element in an array and so on.

Functions can also be used to manipulate structures, for example, you can create a function that initializes a structure, a function that compares two structures, or a function that prints the contents of a structure.

functions, structures, and arrays are all powerful tools for organizing and manipulating data in C and C++ programming languages.

Arrays of Structures

An array of structures is an array that contains elements of a structure type. Each element in the array is a structure variable, and it has the same members as the structure type.

Example:

```
struct point {
   int x;
   int y;
};
struct point points[10]; // Declaring an array of 10 point structures
```

You can initialize the elements of an array of structures when you declare the array or individually later.

Example:

```
struct point points[3] = {{1, 2}, {3, 4}, {5, 6}}; // Initializing all elements when declared
struct point points[3];
points[0] = {1, 2}; // Initializing the first element
points[1] = {3, 4}; // Initializing the second element
points[2] = {5, 6}; // Initializing the third element
```

You can also access the individual members of an array of structures using the array subscript operator [].

Example:points[0].x = 1;

points[0].y = 2;

Using arrays of structures, you can store and manipulate large amounts of data that has a similar structure. For example, you can use an array of structures to store the information of multiple employees in a company, where each element of the array represents an employee and has members for the employee's name, salary, and job title.

Also, you can use loops to iterate over the array of structures and perform operations on each element.

Example:

```
for (int i = 0; i < 10; i++) {
    printf("x: %d, y: %d\n", points[i].x, points[i].y);
}
```

It's also possible to pass array of structures to functions as arguments and return them as results.

Unions with Example

A union is a user-defined data type that allows you to store different data types in the same memory location. Unlike structures, which have separate memory space for each member, all members of a union share the same memory space. The size of a union is the size of its largest member.

Example:

```
union number {
    int i;
    float f;
    double d;
};
```

In this example, we have a union called "number" that has three members: an integer "i", a float "f" and a double "d", all these three members will share the same memory space.

You can initialize a union variable when you declare it, or later when you assign a value to one of its members.

```
union number n = {5}; // initialize when declaring
n.f = 3.14; // change the value of n to 3.14
```

You can also access the individual members of a union using the dot operator (.)

```
printf("The value of n is %f\n", n.f);
```

However, since all members share the same memory space, you can only have one member active at a time. If you assign a value to one member, the previous value stored in the other members will be overwritten.

For example, if you set the value of the "f" member to 3.14, the value of the "i" member will be overwritten with an undefined value.

Unions are mainly used when you want to save memory space, or when you want to manipulate data in different ways depending on its type. For example, you can use a union to store a value that can be interpreted as either an integer or a float, and then switch between the two depending on the needs of your program.

Q: What is a structure in C?

A: In C, a structure is a user-defined data type that can hold different types of data, known as members. A structure is defined using the "struct" keyword, followed by a name, and a list of members enclosed in curly braces.

For example, the following code defines a structure called "Person" that has three members: a name, an age, and a salary:

```
struct Person {
   char name[20];
   int age;
   float salary;
};
```

Q: How can we create and access structure members in C?

A: After defining a structure, you can create a variable of that structure type and access its members using the dot operator (.).

For example, the following code creates a variable called "employee" of the "Person" structure and assigns values to its members:

```
struct Person employee;
```

```
strcpy(employee.name, "John Smith");

employee.age = 30;

employee.salary = 50000.0;
```

You can also create and initialize a structure variable at the same time using the following syntax:

Example

```
struct Person employee = {"John Smith", 30, 50000.0};
```

You can also access structure members using pointers and the arrow operator (->). For example, the following code creates a pointer to a "Person" structure and accesses its members:

```
struct Person *ptr = &employee;

printf("Name: %s\n", ptr->name);

printf("Age: %d\n", ptr->age);

printf("Salary: %f\n", ptr->salary);
```

It's worth noting that the structure members are stored in contiguous memory locations and each member has its own memory location.

Q: What is a union in C?

A: In C, a union is a user-defined data type that can hold different types of data, known as members. A union is similar to a structure, but unlike a structure, all members of a union share the same memory location, meaning that a union can only hold the value of one member at a time. The size of a union is the size of its largest member.

For example, the following code defines a union called "Data" that has three members: an integer, a float, and a character:

```
union Data {

   int i;

   float f;

   char c;
```

};

Q: **How can we create and access union members in C?**

A: After defining a union, you can create a variable of that union type and access its members using the dot operator (.).

For example, the following code creates a variable called "data" of the "Data" union and assigns a value to its integer member:

Example

union Data data;

data.i = 5;

You can also access union members using pointers and the arrow operator (->). For example, the following code creates a pointer to a "Data" union and accesses its integer member:

Example

union Data *ptr = &data;

printf("Integer: %d\n", ptr->i);

Some Question for practice

1. What is the difference between a structure and a union in C?
2. How do you define a structure in C?
3. What is the purpose of a structure tag in C?
4. How do you access members of a structure in C?
5. What is the difference between a structure variable and a structure pointer in C?
6. How do you initialize a structure in C?
7. What is the purpose of the typedef keyword when used with structures in C?
8. How do you pass a structure to a function in C?
9. What is the difference between a nested structure and a structure within a union in C?
10. How do you use an array of structures in C?
11. How do you compare two structures in C?
12. What is the difference between a struct and a class in C++?

13. How do you use a union to save memory in C?
14. What is the purpose of the . (dot) operator when used with structures in C?
15. How do you use pointers to access members of a structure in C?
16. What is the difference between a packed structure and an unpacked structure in C?
17. How do you use a structure within a structure in C?
18. What is the purpose of the -> (arrow) operator when used with structures in C?
19. How do you use a union to access different data types in C?