SNS COLLEGE OF TECHNOLOGY

Coimbatore – 35

An Autonomous Institution

Accredited by NBA – AICTE and Accredited by NAAC – UGC with 'A+' Grade

Approved by AICTE, New Delhi & Affiliated to Anna University, Chennai

DEPARTMENT OF AI & ML

PROGRAMMING FOR PROBLEM SOLVING

I YEAR - I SEM

UNIT V – STRUCTURES AND UNIONS

TOPIC 3 – UNIONS

Introduction

- Imagine an e-commerce company that wants to keep track of all products that it wants to sell. At the most basic level, it would like to keep track of the details of the products it's selling and shipping.
- Each product contains defined properties like *weight*, *dimensions* and *price*. Furthermore, to store details of each product, memory space is required in the computer systems at every warehouse of the company across the country or the world.
- When we consider the size and numbers at which a multinational e-commerce country operates, it becomes clear that the amount of memory space needed to store the details of each product needs to be optimized without compromising the integrity of the data.
- The concept of unions takes shape to help with situations such as these, where related data needs to be stored in a memory-optimized way.
- As mentioned above, unions are user-defined data type that allows many different data types to be stored in the same memory region.
- This essentially means that one data member can have its data stored in the memory at a time, and changing the value of any union member affects the data stored in the memory. Let us now look at how unions can be used in C.

Union

Union can be defined as a user-defined data type which is a collection of different variables of different data types in the same memory location.

The union can also be defined as many members, but only one member can contain a value at a particular point in time.

Union is a user-defined data type, but unlike structures, they share the same memory location.

Syntax:

```
union union_name
{
union_members;
}instance;
```

```
union u
{
char s[5];
int num;
}u1;
```

- ✓ The variable u1 will be allocated memory to accommodate the largest member of the union.
- ✓ In this example, it will be allocated 5 bytes.
- ✓ Only one member can be stored at a time.
- ✓ We can store either array s or num.
- ✓ Both do not exist simultaneously.
- ✓ To calculate the size of the union, the sizeof operator is used. For the example above, sizeof(u1) will be 5.

Accessing Members of the Union

Union members can be accessed using the dot operator i.e.,
union_variable.member

If we have a pointer to the union (similar to the pointer to a structure), the members are accessed using the ->operator.
union_pointer->member

Example:
union u *ptr=&u1;

Accessing members:
u1.s ptr->s
u1.num ptr->num

Initializing a Union

Since only one member of the union can be used at a time, only the first member can be initialized during declaration.

Example:
union u
{
char s[5];
int num;
};
union u var={“ABC”};

Union within a structure

A union can be nested within another union or a structure.

Consider the example: Suppose we wish to store information about an employee-id, name, and type. The type can be **“F” for a full-time employee** or **“P” for a part-time employee**. If the type is **‘F’, store the monthly_salary**. If the type is **‘P’, store the hours_worked**.

The structure and union declarations will be:

```
struct employee
{
char name[20];
int id;
char type;
union info
{
int monthly_salary;
int hours_worked;
}details;
}e1,e2;
```

```
int main() {
strcpy(e1.name,”ABC”);
e1.id=456;
e1.type=’F’;
e1.details.monthly_salary=1500;

strcpy(e2.name,”LMN”);
e2.id=458;
e2.type=’P’;
e2.details.hours_worked=15;
return 0;  }
```

Example of Union

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

union doctor
{
    int id;
    char name[40];
    char department[40];
} doc2;

int main()
{
    union doctor doc1;

    doc1.id = 2234;
    strcpy(doc1.name, "Ravi Krishna");
    strcpy(doc1.department, "Radiology");

    printf("Record 1 details: \n");
    printf("%d \n", doc1.id);
    printf("%s \n", doc1.name);
    printf("%s \n", doc1.department);

    printf("\n");
    printf("Record 2 details: \n");
    doc2.id = 7593;
    printf("%d \n", doc2.id);
    strcpy(doc2.name, "Shreya Srinivas");
    printf("%s \n", doc2.name);
    strcpy(doc2.department, "Inpatient Service");
    printf("%s \n", doc2.department);


    return 0;
}
```

Output

```
Record 1 details:
1768186194
Radiology
Radiology

Record 2 details:
7593
Shreya Srinivas
Inpatient Service
```

Explanation

- We have taken both union and structure in the given example program to understand how values are stored in union variables.
- In record 1, First, the memory of union doc1 contains the value 2234 corresponding to the int data type. Next, when the union member doc1.name has been assigned the value 'Ravi Krishna', the memory location name is now doc1.name, and the value stored in this location is 'Ravi Krishna' corresponding to the char[] data type.
- Next, the union member doc1.department has been assigned the value 'Radiology'. The memory location has been changed to doc1.department and the value to 'Radiology', also corresponding to the char[] data type. Keep in mind unions can hold only one member at a time.
- Therefore, the values in the shared memory are constantly being replaced every time a new member is assigned a value. Thus when the value in doc1.id is printed, a garbage value appears in the Output as the shared memory is held by the char[] datatype. Also, the value printed by name and department is the same as they are of char[] type.
- The values of Record 2 appear correctly at the Output. This is because the union members' values are printed before their address and values are changed.

Applications of Unions

- Unions are used when mutually exclusive data members have to share the memory in places where memory space is at a premium, such as embedded systems.
- Unions are also used when the program needs direct memory access.

To further understand their applications, let's take up an example:

Imagine a two-wheeler dealership that sells motorbikes and bicycles. The dealership owner would like to keep track of the inventory of the items in his establishment and store the relevant information in a computer system. The motorcycle has a price, engine size, and mileage, whereas the bicycle has the properties of color and price. The property of price is common to both items. The shop owner would now like to store them as records.

Similarities between Structure & Unions

1.Both are user-defined data types used to store data of different types as a single unit.

2.Their members can be objects of any type, including other structures and unions or arrays. A member can also consist of a bit field.

3.Both structures and unions support only assignment = and sizeof operators. The two structures or unions in the assignment must have the same members and member types.

4.A structure or a union can be passed by value to functions and returned by value by functions. The argument must have the same type as the function parameter. A structure or union is passed by value just like a scalar variable as a corresponding parameter.

5.‘.’ operator or selection operator, which has one of the highest precedences, is used for accessing member variables inside both the user-defined datatypes.

Difference between Structures & Unions

	STRUCTURE	UNION
Keyword	The keyword **struct** is used to define a structure	The keyword **union** is used to define a union.
Size	When a variable is associated with a structure, the compiler allocates the memory for each member. The size of structure is **greater than or equal to the sum of sizes of its members.**	when a variable is associated with a union, the compiler allocates the memory by consideringthe size of the largest memory. So, size of **union is equal to the size of largest member.**
Memory	Each member within a structure is assigned unique storage area of location.	Memory allocated is shared by individual members of union.
Value Altering	Altering the value of a member will not affect other members of the structure.	Altering the value of any of the member will alter other member values.
Accessing members	Individual member can be accessed at a time.	Only one member can be accessed at a time.
Initialization of Members	Several members of a structure can initialize at once.	Only the first member of a union can be initialized.