Department of Computer Science and Engineering, PES University, Bangalore, India

Lecture Notes
Problem Solving With C
UE24CS151B

Lecture #17
Unions in C

By,

Prof. Sindhu R Pai,

Theory Anchor, Feb-May, 2025

Assistant Professor

Dept. of CSE, PESU

Many Thanks to

Dr. Shylaja S S (Director, CCBD and CDSAML Research Center, PES University)

Prof. Nitin V Pujari (Dean, Internal Quality Assurance Cell, PES University)

Unit #: 3

Unit Name: Text Processing and User-Defined Types

Topic: Unions in C

Course objectives: The objective(s) of this course is to make students

- Acquire knowledge on how to solve relevant and logical problems using computing Machine.
- Map algorithmic solutions to relevant features of C programming language constructs.
- Gain knowledge about C constructs and its associated ecosystem.
- Appreciate and gain knowledge about the issues with C Standards and it's respective behaviours.

Course outcomes: At the end of the course, the student will be able to:

- Understand and Apply algorithmic solutions to counting problems using appropriate C Constructs.
- Understand, Analyze and Apply sorting and Searching techniques.
- Understand, Analyze and Apply text processing and string manipulation methods using Arrays, Pointers and functions.
- Understand user defined type creation and implement the same using C structures, unions and other ways by reading and storing the data in secondary systems which are portable.

Sindhu R Pai

Theory Anchor, Feb - May, 2025

Dept. of CSE,

PES University

Introduction

A Union is a user-defined datatype in C programming language. It is a collection of **variables of different datatypes in the same memory location**. We can define a union with many members, but at a given point of time, only one member can contain a value. Union unlike structures, share the same memory location. Using Union in C will save Memory Space in a given context. C unions allow data members which **are mutually exclusive to share the same memory.**

Syntax:

```
union union_name {
        data_type1 member1;
        data_type2 member2;
        ...
        data_typeN memberN;
};
```

Note:

- The **size of a union** is at least equal to the size of its **largest member**.
- Updating one member will **overwrite** the other/s, since they all occupy the same memory location.

The memory occupied by a union will be large enough to hold the largest member of the union. So, the size of a union is the size of the biggest component. Compilers typically add padding to align the union size to the nearest multiple of 4 or 8 (for efficient memory access). At a given point in time, only one can exist. All the fields overlap and they have the **same offset: 0**

Example:

```
        union Data {
                int i;
                float f;
                char str[20];
        };
```

Characteristics of Unions in C

1. All members of a **union share the same memory location** and only **one member holds a valid value at any time.**
2. The **size of a union** is **equal to the size of its largest member**, possibly including padding for alignment.
3. Useful when different members are used **one at a time**, as it **saves memory** compared to structures.
4. Assigning a value to one member **destroys the previous content** stored in another member.
5. **Type Reinterpretation (Type Punning)**: You can use unions to **reinterpret memory** (e.g., access the same bytes as both float and int).
6. Unions can be **passed to and returned from functions**, like structs.
7. A union can **contain structs or other unions** as members.
8. Accessing multiple members at once leads to **undefined behavior.**
9. **Storage classes** apply **to variables, not to members within the union.**

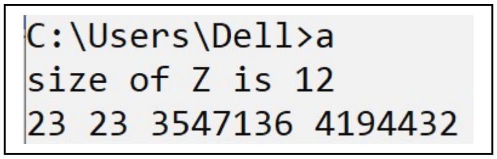
Coding Example_1: Printing all members of a union

```
#include <stdio.h>
#include <string.h>
int main() {
    union Data d;

    // Assign to int
    d.i = 100;     printf("After assigning int:\n");
    printf("d.i = %d\n", d.i);    printf("d.f = %f\n", d.f);
    printf("d.str = %s\n\n", d.str);  // May print garbage

    // Assign to float
    d.f = 3.14;     printf("After assigning float:\n");
    printf("d.i = %d\n", d.i);        // May print garbage
    printf("d.f = %f\n", d.f);
    printf("d.str = %s\n\n", d.str);  // May print garbage
```

```
 // Assign to string
   strcpy(d.str, "Hello");
   printf("After assigning string:\n");
   printf("d.i = %d\n", d.i);        // May print garbage
   printf("d.f = %f\n", d.f);        // May print garbage
   printf("d.str = %s\n", d.str);
   return 0;
}
```

Coding Example_2: Size of union(assuming the size of int is 4 bytes) and accessing the union members

```
#include<stdio.h>
union Z
{        int a;   int b[3];        };
int main()
{
        union Z z;      z.a = 23;
        printf("size of Z is %d\n",sizeof(z));
        printf("%d %d %d",z.b[0], z.b[1],z.b[2]);
        return 0;
}
```



```
C:\Users\Dell>a
size of Z is 12
23 23 3547136 4194432
```

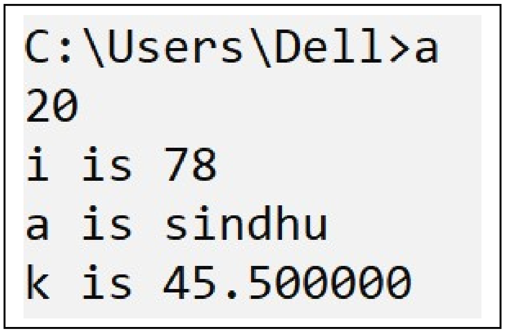
Coding Example_3: All the fields overlap and they have the same offset: 0 in union

```
#include<stdio.h>
#include<stddef.h>            // offsetof function is from this header file
union A
{        int x;   int y;   int z;
};
struct B
{        int x;   int y;   int z;
};
```

```
int main()
{
        printf("%lu\n",offsetof(union A,y));  // 0
        printf("%lu\n",offsetof(struct B,y));    // 4
        printf("%lu\n",offsetof(struct B,z));    // 8
        // assumption int occupies four bytes
}
```

Coding Example_4: Usage of anonymous union inside a union

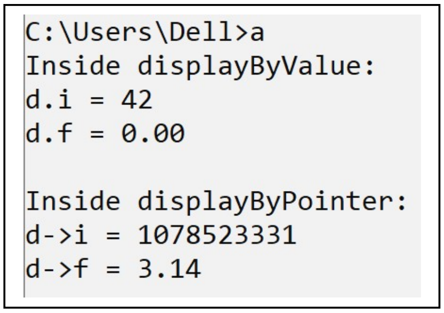
```
#include<stdio.h>
union test
{
        int i;
        union
        {       char a[20];             float k;        };
};
int main()
{
        printf("%lu",sizeof(union test));
        union test t;
        t.i=78;   // One member at a time from union
        printf("\ni is %d\n",t.i);
        strcpy(t.a,"sindhu");
        printf("a is %s\n",t.a);
        t.k=45.5;
        printf("k is %f",t.k);
        return 0;
}
```



```
C:\Users\Dell>a
20
i is 78
a is sindhu
k is 45.500000
```

Coding Example_5: Passing a union to a function using two ways.

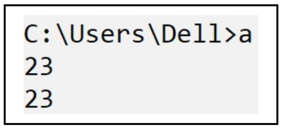
```
#include <stdio.h>
union Data {   int i;     float f;   };
void displayByValue(union Data d) {
   printf("Inside displayByValue:\n");
   printf("d.i = %d\n", d.i);
   printf("d.f = %.2f\n\n", d.f);
}
void displayByPointer(union Data *d) {
   printf("Inside displayByPointer:\n");
   printf("d->i = %d\n", d->i);
   printf("d->f = %.2f\n\n", d->f);
}
int main() {
   union Data d;    d.i = 42;    displayByValue(d);
   d.f = 3.14;    displayByPointer(&d);
   return 0;
}
```



```
C:\Users\Dell>a
Inside displayByValue:
d.i = 42
d.f = 0.00

Inside displayByPointer:
d->i = 1078523331
d->f = 3.14
```

Coding Example_6: Reading and displaying the union member

```
void read(union Data *d)
{         scanf("%d", &(d->i));         }
void display(union Data d)
{         printf("%d", (d.i));            }
int main() {
          union Data d;  read(&d);        display(d);
          return 0;
}
```



```
C:\Users\Dell>a
23
23
```

Unions: Code smart, save memory!