Andhra Pradesh State Skill Development Corporation

Skill AP
APSSDC

eSkill AP
Learn Anytime Anywhere
Andhra Pradesh State Skill Development Corporation

Programming in C

Unions

UNION

A union is a special data type available in C that allows storing different data types in the same memory location. You can define a union with many members, but only one member can contain a value at any given time. Unions provide an efficient way of using the same memory location for multiple-purpose.

Unions are quite similar to structures in C. Like structures, unions are also derived types.

Syntax:

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

Defining a union is as easy as replacing the keyword struct with the keyword union.

Creation of Union:

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

Another way

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

int main(){
  union car car1, car2, *car3;
  return 0;
}
```

In both cases, union variables car1, car2 and union pointer variable car3 of type union car are created.

Accessing elements of a union

Again, the elements of unions can be accessed in a similar manner as structures.

In the above example, suppose you want to access price for union variable car1, it can be accessed as:

```
car1.price
```

Memory allocation:

As seen in the above example, there is a difference in memory allocation between union and structure.
The amount of memory required to store a structure variable is the sum of memory size of all members.

Fig: Memory allocation in case of structure

But, the memory required to store a union variable is the memory required for the largest element of an union.

Fig: Memory allocation in case of union

Example

```
#include
#include

union student
{
        char name[20];
        char subject[20];
        float percentage;
};

int main()
{
   union student record1;
   union student record2;
```

```
    // assigning values to record1 union variable
       strcpy(record1.name, "Raju");
       strcpy(record1.subject, "Maths");
       record1.percentage = 86.50;

       printf("Union record1 values example\n");
       printf(" Name       : %s \n", record1.name);
       printf(" Subject    : %s \n", record1.subject);
       printf(" Percentage : %f \n\n", record1.percentage);

    // assigning values to record2 union variable
       printf("Union record2 values example\n");
       strcpy(record2.name, "Mani");
       printf(" Name       : %s \n", record2.name);

       strcpy(record2.subject, "Physics");
       printf(" Subject    : %s \n", record2.subject);

       record2.percentage = 99.50;
       printf(" Percentage : %f \n", record2.percentage);
       return 0;
}
```

Output:

```
Union record1 values example
 Name       :
 Subject    :
 Percentage : 86.500000

Union record2 values example
 Name       : Mani
 Subject    : Physics
 Percentage : 99.500000
Press any key to continue . . .
```

Difference between union and structure

Though unions are similar to structure in so many ways, the difference between them is crucial to understand.

The primary difference can be demonstrated by this **example**:

```
#include
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", sizeof(uJob));
   printf("\nsize of structure = %d \n", sizeof(sJob));
   return 0;
}
```

Output:

```
size of union = 32
size of structure = 40
Press any key to continue . . .
```

Difference Between Structure and Union

C Structure	C Union
Structure allocates storage space for all its members separately.	Union allocates one common storage space for all its members. Union finds that which of its member needs high storage space over other members and allocates that much space
Structure occupies higher memory space.	Union occupies lower memory space over structure.
We can access all members of structure at a time.	We can access only one member of union at a time.
Structure example: struct student { int mark; char name[6]; double average;	Union example: union student { int mark; char name[6]; double average;

};	};
For above structure, memory allocation will be like below. int mark – 2B char name[6] – 6B double average – 8B Total memory allocation = 2+6+8 = 16 Bytes	For above union, only 8 bytes of memory will be allocated since double data type will occupy maximum space of memory over other data types. Total memo

**** THE END ****