Programming and Problem Solving through C Language
O Level / A Level

Chapter - 8 : Structures and Unions

Unions

- A union is declared and used in the same ways that a structure.
- Unions are defined and declared in the same fashion as structures.
- In unions, all the members share the space which is according to the space requirement of the largest member.
- The union can hold only one value at a time.
- A union can be initialized on its declaration.
- Because only one member can be used at a time, only one can be initialized.
- To avoid confusion, only the first member of the union can be initialized.

Defining of Union

- A union has to defined, before it can be used.
- The syntax of defining a structure is

```
union <union_name>
{
        <data_type>  <variable_name>;
        <data_type> <variable_name>;
       ............
        <data_type> <variable_name>;
} ;
```

Example

To define a simple union of a char variable and an integer variable

```
union shared
{       char c;
        int i;
} ;
```

This union, **shared**, can be used to create instances of a union that can hold either a character value(c) or an integer value(i).

Union Data Type

- A union is a user defined data type like structure.
- The union groups logically related variables into a single unit.
- The union data type allocates the space equal to space needed to hold the largest data member of union.
- The union allows different types of variable to share same space in memory.
- There is no other difference between structure and union than internal difference.
- The method to declare, use and access the union is same as structure.

Accessing Union Members

- Individual union members can be used in the same way that structure members can be used--by using the member dot operator (.).
- Only one union member should be accessed at a time, as different variable share same space in memory.

Example

```
union shared
{ char c;
  int i;
};

shared.c ='a';
shared.d=1;
```

In this case, value assignment to member variable (d) overwrites the member variable(c).

Difference between Structures & Union

- The memory occupied by structure variable is the sum of sizes of all the members but memory occupied by union variable is equal to space hold by the largest data member of a union.
- In the structure all the members can be accessed at any point of time but in union only one of union member can be accessed at any given time.

Example

```
#include<stdio.h>

union job
{ char name[32];
  float salary;
  int worker_no;
} u;

struct job1
{ char name[32];
  float salary;
  int worker_no;
} s;

void main( )
{ printf("size of union = %d", sizeof(u));
  printf("\nsize of structure = %d", sizeof(s));
}
```

output

```
size of union = 32
size of structure = 40
```

Difference between Structure and Union

Structure	Union
1. It can be defined using **struct** keyword.	It can be defined using a **union** keyword.
2. Every member within structure is assigned a unique memory location.	In union, a memory location is shared by all the data members.
3. Changing the value of one data member will not affect other data members in structure.	Changing the value of one data member will change the value of other data members in union.
4. It allows initializing several members at once.	It allows initializing only the first member of union.
5. The total size of the structure is the sum of the size of every data member.	The total size of the union is the size of the largest data member.
6. It is used for storing various data types.	It is used for storing one of the many data types that are available.
7. It reserves space for each and every member separately.	It reserves space for a member having the highest size.
8. Any member can be retrieve at a time.	Only one member can be retrieve at a time.
9. It allows dynamic array as member.	It does not allows dynamic array as member.

Nested Structure and Union

- A structure can have nested union as a member.
- A union can have a nested structure as a member.
- A union can also have a nested union as a member.

Example.

```
union shared
{ char c;
  int i;
};

struct ABC
{    int x;
     union shared y;
};
```

```
struct shared
{ char c;
  int i;
};

union ABC
{    int x;
     struct shared y;
};
```