UNIONS

Declaration of union must start with the keyword ***union*** followed by the union name and union's member variables are declared within braces.

Syntax for declaring union

```
union union-name
{
        datatype var1;
        datatype var2;
        - - - - - - - - - -
        - - - - - - - - - -
        datatype varN;
};
```

We have to create an object of union to access its members. Object is a variable of type union. Union members are accessed using the dot operator(.) between union's object and union's member name.

Syntax for creating object of union

```
union union-name obj;
```

Example for creating object & accessing union members

```
#include<iostream>

union Employee
{
```

```
          int Id;
          char Name[25];
          int Age;
          long Salary;
     };

     void main()
     {

          Employee E;

               cout << "\nEnter Employee Id : ";
               cin >> E.Id;

               cout << "\nEnter Employee Name : ";
               cin >> E.Name;

               cout << "\nEnter Employee Age : ";
               cin >> E.Age;

               cout << "\nEnter Employee Salary : ";
               cin >> E.Salary;

               cout << "\n\nEmployee Id : " << E.Id;
               cout << "\nEmployee Name : " << E.Name;
               cout << "\nEmployee Age : " << E.Age;
               cout << "\nEmployee Salary : " << E.Salary;



     }
```

Output :

```
Enter Employee Id : 1
Enter Employee Name : Kumar
Enter Employee Age : 29
Enter Employee Salary : 45000

Employee Id : -20536
Employee Name : ?$?$  ?
Employee Age : -20536
Employee Salary : 45000
```

In the above example, we can see that values of Id, Name and Age members of union got corrupted because final value assigned to the variable has occupied the memory location and this is the reason that the value of salary member is getting printed very well. Now let's look into the same example once again where we will use one variable at a time which is the main purpose of having union:

```
#include<iostream>

union Employee
{
    int Id;
    char Name[25];
    int Age;
    long Salary;
};

void main()
{

    Employee E;

        cout << "\nEnter Employee Id : ";
```

```
            cin >> E.Id;
            cout << "Employee Id : " << E.Id;

            cout << "\n\nEnter Employee Name : ";
            cin >> E.Name;
            cout << "Employee Name : " << E.Name;

            cout << "\n\nEnter Employee Age : ";
            cin >> E.Age;
            cout << "Employee Age : " << E.Age;

            cout << "\n\nEnter Employee Salary : ";
            cin >> E.Salary;
            cout << "Employee Salary : " << E.Salary;

    }
```

Output :

```
        Enter Employee Id : 1
        Employee Id : 1

        Enter Employee Name : Kumar
        Employee Name : Kumar

        Enter Employee Age : 29
        Employee Age : 29

        Enter Employee Salary : 45000
        Employee Salary : 45000
```

Here, all the members are getting printed very well because one member is being used at a time

DIFFERENCES BETWEEN C AND C++

Difference between C and C++

Similarities between C and C++ are:

- Both the languages have a similar syntax.
- Code structure of both the languages are same.
- The compilation of both the languages is similar.
- They share the same basic syntax. Nearly all of C's operators and keywords are also present in C++ and do the same thing.
- C++ has a slightly extended grammar than C, but the basic grammer is the same.
- Basic memory model of both is very close to the hardware.
- Same notions of stack, heap, file-scope and static variables are present in both the languages.

C++ can be said a superset of C. Major added features in C++ are Object-Oriented Programming, Exception Handling and rich C++ Library.

Below is the table of differences between C and C++:

C	C++
C was developed by Dennis Ritchie between the year 1969 and 1973 at AT&T Bell Labs.	C++ was developed by Bjarne Stroustrup in 1979.
C does no support polymorphism, encapsulation, and inheritance which means that C does not support object oriented programming.	C++ supports polymorphism, encapsulation, and inheritance because it is an object oriented programming language.

C	C++
C is a subset of C++.	C++ is a superset of C.
C contains 32 keywords.	C++ contains 52 keywords.
For the development of code, C supports procedural programming.	C++ is known as hybrid language because C++ supports both procedural and object oriented programming paradigms.
Data and functions are separated in C because it is a procedural programming language.	Data and functions are encapsulated together in form of an object in C++.
C does not support information hiding.	Data is hidden by the Encapsulation to ensure that data structures and operators are used as intended.
Built-in data types is supported in C.	Built-in & user-defined data types is supported in C++.
C is a function driven language because C is a procedural programming language.	C++ is an object driven language because it is an object oriented programming.
Function and operator overloading is not supported in C.	Function and operator overloading is supported by C++.
C is a function-driven language.	C++ is an object-driven language

C	C++
Functions in C are not defined inside structures.	Functions can be used inside a structure in C++.
Namespace features are not present inside the C.	Namespace is used by C++, which avoid name collisions.
Header file used by C is stdio.h.	Header file used by C++ is iostream.h.
Reference variables are not supported by C.	Reference variables are supported by C++.
Virtual and friend functions are not supported by C.	Virtual and friend functions are supported by C++.
C does not support inheritance.	C++ supports inheritance.
Instead of focusing on data, C focuses on method or process.	C++ focuses on data instead of focusing on method or procedure.
C provides malloc() and calloc() functions for dynamic memory allocation, and free() for memory de-allocation.	C++ provides new operator for memory allocation and delete operator for memory de-allocation.
Direct support for exception handling is not supported by C.	Exception handling is supported by C++.
scanf() and printf() functions are used for input/output in C.	cin and cout are used for input/output in C++.

DIFFERENCES BETWEEN STRUCTURES AND CLASS

Structure vs class in C++

In C++, a structure is the same as a class except for a few differences. The most important of them is security. A Structure is not secure and cannot hide its implementation details from the end user while a class is secure and can hide its programming and designing details. Following are the points that expound on this difference:

1) Members of a class are private by default and members of a struct are public by default.
For example program 1 fails in compilation and program 2 works fine.

```
// Program 1

#include <stdio.h>


class Test {

    int x; // x is private

};

int main()

{

  Test t;

  t.x = 20; // compiler error because x is private

  getchar();

  return 0;

}
```

```
// Program 2

#include <stdio.h>


struct Test {

    int x; // x is public

};
```

```
int main()
{
  Test t;
  t.x = 20; // works fine because x is public
  getchar();
  return 0;
}
```

2) When deriving a struct from a class/struct, default access-specifier for a base class/struct is public. And when deriving a class, default access specifier is private. For example program 3 fails in compilation and program 4 works fine.

```
// Program 3
#include <stdio.h>

class Base {
public:
    int x;
};

class Derived : Base { }; // is equilalent to class Derived : private Base {}

int main()
{
  Derived d;
  d.x = 20; // compiler error becuase inheritance is private
  getchar();
  return 0;
}
```

```
// Program 4
#include <stdio.h>
```

```
class Base {
public:
    int x;
};

struct Derived : Base { }; // is equilalent to struct Derived : public Base {}

int main()
{
  Derived d;
  d.x = 20; // works fine becuase inheritance is public
  getchar();
  return 0;
}
```