Abstraction

Data Abstraction in C++

Data Abstraction is a process of providing only the **essential details to the outside world and hiding the internal details**, i.e., representing only the essential details in the program.

Data Abstraction is a programming technique that depends on the seperation of the interface and implementation details of the program.

Let's take one real life example of a TV, which you can turn on and off, change the channel, adjust the volume, and add external components such as speakers, VCRs, and DVD players, BUT you do not know its internal details, that is, you do not know how it receives signals over the air or through a cable, how it translates them, and finally displays them on the screen.

Thus, we can say a television clearly separates its internal implementation from its external interface and you can play with its interfaces like the power button, channel changer, and volume control without having any knowledge of its internals.

C++ provides a great level of abstraction. For example, **pow()** function is used to calculate the power of a number without knowing the **algorithm the function** follows.

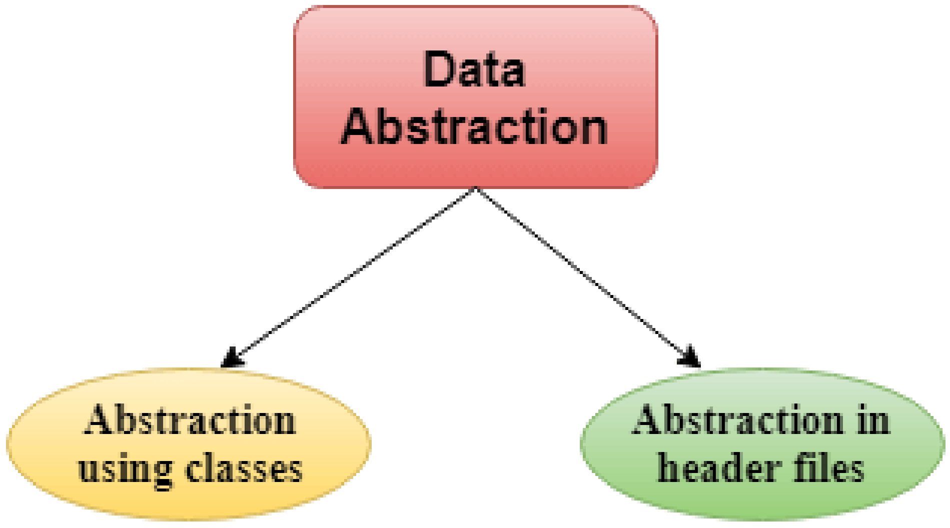
In C++ program if we implement class with **private and public members** then it is an example of data abstraction.

OR

Abstraction is one of the feature of Object Oriented Programming, where you **show only relevant details to the user and hide irrelevant details**. For example, when you send an email to someone you just click send and you get the success message, what actually happens when you click send, how data is transmitted over network to the recipient is hidden from you (because it is irrelevant to you).

Data Abstraction can be achieved in two ways:

1. Abstraction using classes
2. Abstraction in header files.



Data Abstraction in C++

Abstraction using classes:

An abstraction can be achieved using classes. A class is used to group all the data members and member functions into a single unit by using the access specifiers. A class has the responsibility to determine which data member is to be visible outside and which is not.

Let's see a simple example of data abstraction using classes.

```
#include <iostream>
using namespace std;
 class Sum
{
private:
int x, y, z; // private variables
public:
void add()
{
cout<<"Enter two numbers: ";
cin>>x>>y;
z= x+y;
cout<<"Sum of two number is: "<<z<<endl;
}
};
int main()
{
Sum sm;
sm.add();
return 0;
}
```

Output:

```
Enter two numbers:
3
6
Sum of two number is: 9
```

In the above example, abstraction is achieved using classes. A class 'Sum' contains the private members x, y and z are only accessible by the member functions of the class.

Abstraction in header files:

An another type of abstraction is header file. For example, **pow()** function available is used to **calculate the power of a number without actually knowing which algorithm function** uses to calculate the power. Thus, we can say that **header files hides all the implementation details from the user.**

Access Specifiers Implement Abstraction:

Public specifier:

When the members are declared as public, members can be accessed anywhere from the program.

Private specifier:

When the members are declared as private, members can only be accessed only by the member functions of the class.

Let's see a simple example of abstraction in header files.

// program to calculate the power of a number.

```
#include <iostream>
#include<math.h>
using namespace std;
int main()
{
 int result = pow(4,3);        // 4 is baase & 3 is power
  cout << "Cube of n is : " <<result<<endl;
   return 0;
}
```

Output:

```
Cube of n is : 64
```

In the above example, pow() function is used to calculate 4 raised to the power 3. The pow() function is present in the math.h header file in which all the implementation details of the pow() function is hidden.