**Data abstraction**

Data abstraction is one of the most essential and important features of object-oriented programming in C++. Abstraction means displaying only essential information and hiding the details. Data abstraction refers to providing only essential information about the data to the outside world, hiding the background details or implementation.

Consider a real-life example of a person driving a car. The person only knows that using the accelerators will increase the speed of car or applying brakes will stop the car but he does not know about how on pressing accelerator the speed is actually increasing, he does not know about the inner mechanism of the car or the implementation of accelerator, brakes etc in the car. This is what abstraction is.  
 **Abstraction using Classes:** We can implement Abstraction in C++ using classes. Class helps us to group data members and member functions using available access specifiers. A Class can decide which data member will be visible to outside world and which is not.

**Abstraction in Header files:**One more type of abstraction in C++ can be header files. For example, consider the pow() method present in math.h header file. Whenever we need to calculate power of a number, we simply call the function pow() present in the math.h header file and pass the numbers as arguments without knowing the underlying algorithm according to which the function is actually calculating power of numbers.

**Abstraction using access specifiers**

Access specifiers are the main pillar of implementing abstraction in C++. We can use access specifiers to enforce restrictions on class members. For example:

* Members declared as **public**in a class, can be accessed from anywhere in the program.
* Members declared as **private**in a class, can be accessed only from within the class. They are not allowed to be accessed from any part of code outside the class.

We can easily implement abstraction using the above two features provided by access specifiers. Say, the members that defines the internal implementation can be marked as private in a class. And the important information needed to be given to the outside world can be marked as public. And these public members can access the private members as they are inside the class.

**Example**:

|  |
| --- |
| #include <iostream>  **class** implementAbstraction  {  **private**:  **int** a, b;    **public**:            // method to set values of          // private members  **void** set(**int** x, **int** y)          {              a = x;              b = y;          }    **void** display()          {              cout<<"a = " <<a << endl;              cout<<"b = " << b << endl;          }  };    **int** main()  {      implementAbstraction obj;      obj.set(10, 20);      obj.display();  **return** 0;  } |

Output:

a = 10

b = 20

You can see in the above program we are not allowed to access the variables a and b directly, however one can call the function set() to set the values in a and b and the function display() to display the values of a and b.

**Advantages of Data Abstraction**:

* Helps the user to avoid writing the low-level code
* Avoids code duplication and increases reusability.
* Can change internal implementation of class independently without affecting the user.
* Helps to increase security of an application or program as only important details are provided to the user.