# Abstraction

Course Code: CSC1102 &1103

Course Title: Introduction to Programming

**Dept. of Computer Science**
**Faculty of Science and Technology**

| | | | | | |
|---|---|---|---|---|---|
| **Lecturer No:** | | **Week No:** | | **Semester:** | |
| **Lecturer:** | *MD. MAZID-UL-HAQUE* | | | | |

## Lecture Outline

- Abstraction
- Data Abstraction
- Other ways:
    - * Abstract Class
    - * Abstract Function (Pure Virtual Method)

## Abstraction

Definition of Abstraction

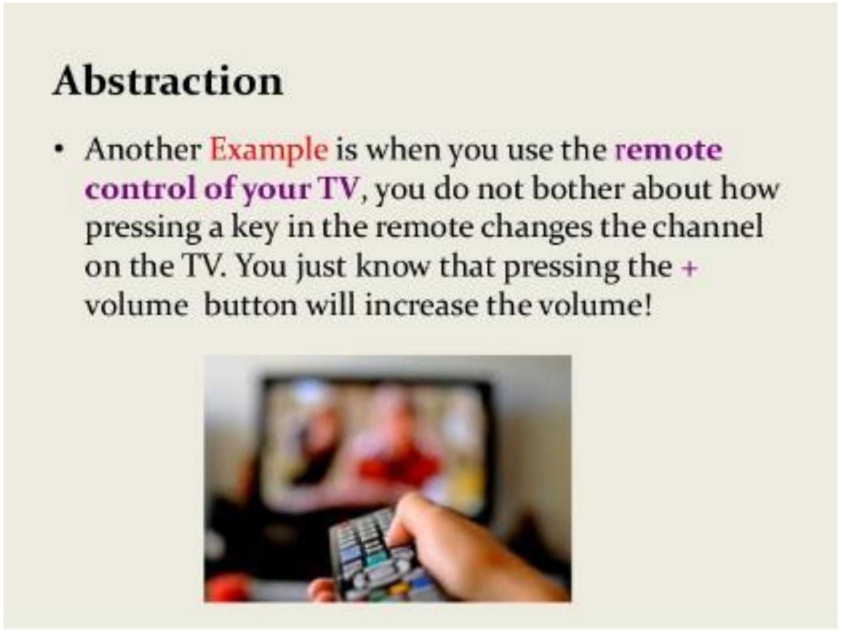
- It means ‘Hiding the Details’.
- It is a process where implementation is hidden and its functionality is shown to user.

## Abstraction

Example of Abstraction

Real Life Example:

ATM
Sitesbay.com
Real Life Example of Abstraction



**Abstraction**

- Another Example is when you use the **remote control of your TV**, you do not bother about how pressing a key in the remote changes the channel on the TV. You just know that pressing the + volume button will increase the volume!

## Abstraction

In programming, this concept of hiding the details is known as abstraction.

We can hide the details of a class or a function.

- Data Abstraction in C++
- Other ways:
  - Abstract Class
  - Abstract Function (In C++ known as Pure Virtual Method)

## Data Abstraction

- What is Data Abstraction?

Data abstraction refers to providing only essential information about the data to the outside world, hiding the background details or implementation.

- How to achieve data abstraction?
  1. Abstraction using classes
  2. Abstraction using header files.

## Data Abstraction

### 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. We can achieve it using **access specifier**.

## Data Abstraction

- Access specifiers

In C++, we have **3 access specifiers**.

1. **Public**: members can be accessed from outside of the class
2. **Private**: members can be accessed from outside on the class. It is only accessible only from the containing class.
3. **Protected**: members cannot be accessed from outside the class, however, they can be accessed in inherited classes.

## Data Abstraction (Example-using class)

```
#include<iostream>
using namespace std;
class implementAbstraction
{
   private: ///Private members
      int a,b;
   public:
   void set(int x, int y ) ///method to set
values of private members
   {
      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;
}
```

User just know that there is two function named set and display to access two data/variables, but will not be able to directly access them.

## Data Abstraction

We can see in the above program we are not allowed to access the variables a and b directly as this is a private member of this class, 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. But a and b can’t be accessed directly from the main method. So, a and b are hidden from the outside of the class. So, this is an abstractions.

## Data Abstraction

- Abstraction using header file

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/logic according to which the function is actually calculating power of numbers.

## Data Abstraction (Example-using header file)

```
#include<iostream>
#include<math.h> ///Imported math header
using namespace std;

int main()
{
   int A;
   int power;
   cout << "Enter a number: " ;
   cin>> A;
   cout << "Enter power: ";
   cin >> power;
   int Power_of_A= pow(A, power);  ///pow()
   cout <<A<<"^"<<power<<"= "<<Power_of_A;
   return 0;
}
```

User knows there is a function named “pow” in the header file, but does not know the detailed implementation of this “pow“ function

## Other ways

## Abstract Function (Pure Virtual Function)

A pure virtual function (or abstract function) in C++ is a virtual function for which we can have implementation, but we must override that function in the derived class, otherwise the derived class will also become abstract class. So, in the base class it (pure virtual function) have no implementation or the detail of the function is hidden on the base class.

## Abstract Function (Pure Virtual Function)(Example)

```
#include<iostream>
using namespace std;

class Base
{
    int x;
    public:
        virtual void fun() = 0; ///Pure virtual
function
                                ///Abstract function

        int getX()
        {
            return x;
        }
};
```

```
/// This class inherits from Base and
implements fun()
class Derived: public Base

{
    int y;
    public:
        void fun() ///Implemented Pure
virtual/Abstract function
        {
            cout << "fun() called";
        }

};
```

## Abstract Class

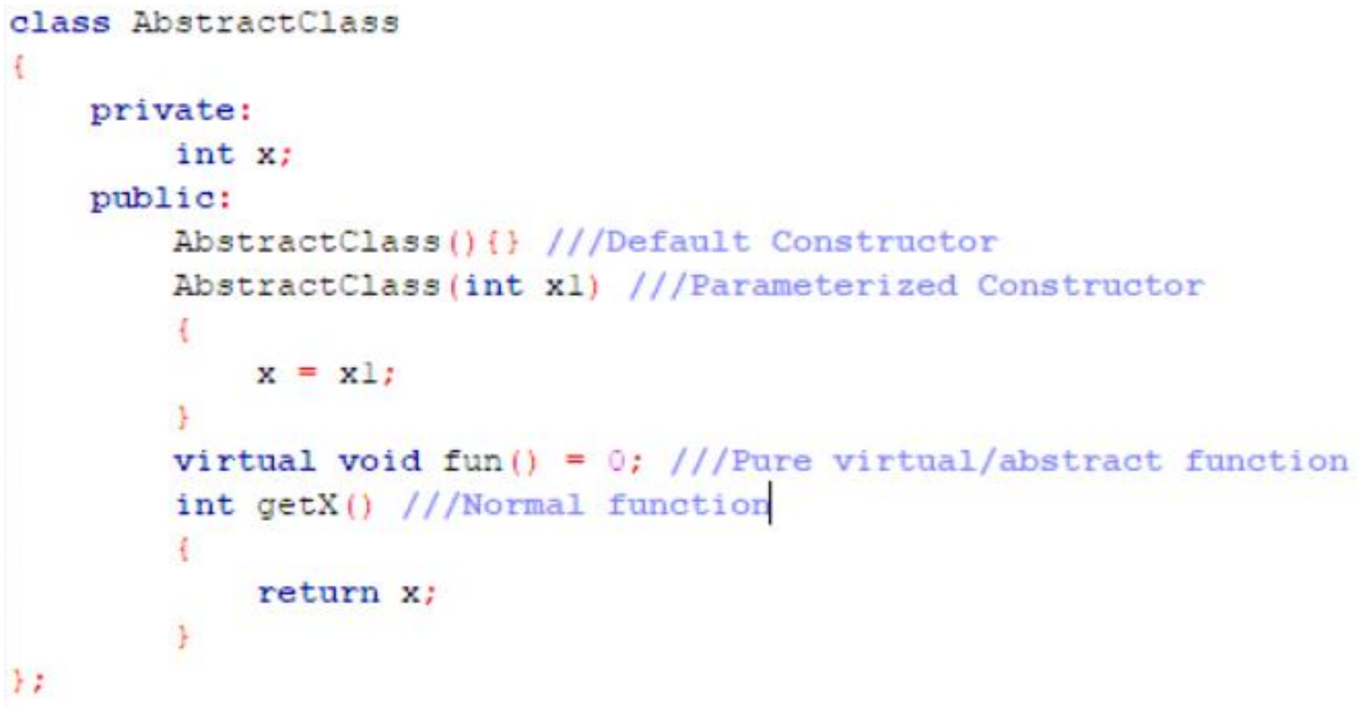
- What is abstract class?

By definition, an **abstract class in C++** is a class that has at least *one* **pure virtual function** (i.e., a function that has no definition).

  - We can’t create instance/object of a abstract class directly. However pointers and references of Abstract class type can be created.

## Abstract Class

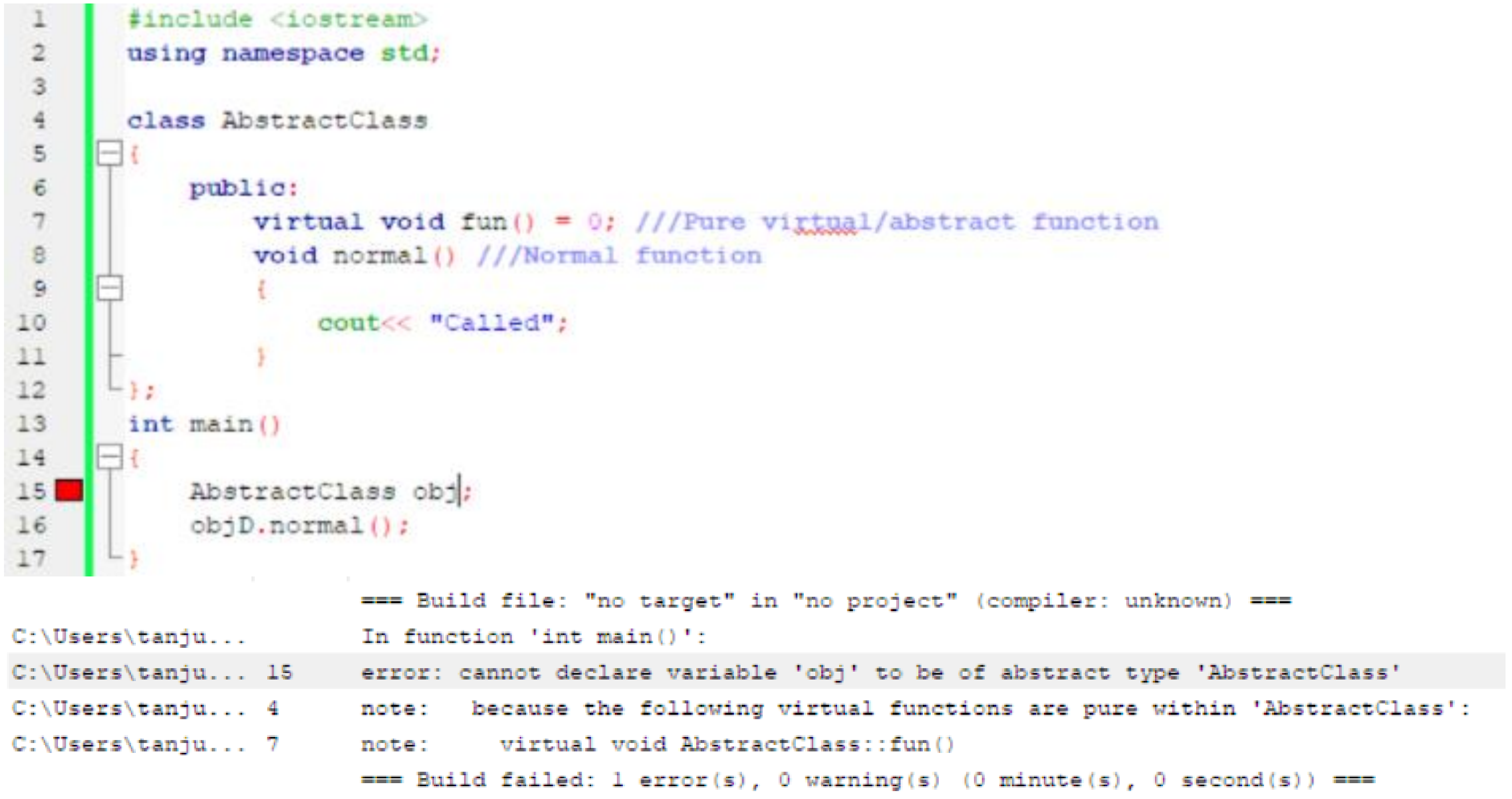
- What an abstract class can contain?
  1. An abstract class may have regular function.
  2. An abstract class must have at least one pure virtual function.
  3. An abstract class may have attributes.
  4. An abstract class may have constructors.



```
class AbstractClass
{
    private:
        int x;
    public:
        AbstractClass(){} ///Default Constructor
        AbstractClass(int x1) ///Parameterized Constructor
        {
            x = x1;
        }
        virtual void fun() = 0; ///Pure virtual/abstract function
        int getX() ///Normal function
        {
            return x;
        }
};
```

## Abstract Class

- We can’t create instance/object of a abstract class.



```
1   #include <iostream>
2   using namespace std;
3
4   class AbstractClass
5   {
6       public:
7           virtual void fun() = 0; ///Pure virtual/abstract function
8           void normal() ///Normal function
9           {
10              cout<< "Called";
11          }
12  };
13  int main()
14  {
15      AbstractClass obj;
16      objD.normal();
17  }
```

```
                          === Build file: "no target" in "no project" (compiler: unknown) ===
C:\Users\tanju...         In function 'int main()':
C:\Users\tanju...  15     error: cannot declare variable 'obj' to be of abstract type 'AbstractClass'
C:\Users\tanju...  4      note:   because the following virtual functions are pure within 'AbstractClass':
C:\Users\tanju...  7      note:     virtual void AbstractClass::fun()
                          === Build failed: 1 error(s), 0 warning(s) (0 minute(s), 0 second(s)) ===
```

## Accessing Abstract Function and Class

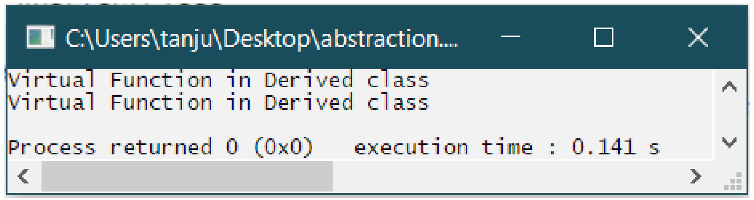
- We can access both abstract function and class by inheritance.
- We have to inherit the abstract class by another class and have to implement the abstract function (pure virtual function) on it.
- Then we can access the abstract function and all the public and protected members of the abstract call from the child class (the class which inherits the abstract class).
- From that we will able to create pointers and references of Abstract class.

## Accessing Abstract Function and Class (Example)

```cpp
#include<iostream>
using namespace std;
class AbstractClass
{
   public:
      virtual void fun() = 0; /// Pure Virtual/abstract Function
};

class DeriverdClass:public AbstractClass
{
   public:
      void fun()///Implemented Pure Virtual/abstract Function
      {
         cout << "Virtual Function in Derived class\n";
      }
};

int main() {
   AbstractClass *b; ///Created pointer of Abstract class
   DeriverdClass dobj;
   dobj.fun();
   b = &dobj; ///Created reference of Abstract class
   b->fun();
}
```



```
Virtual Function in Derived class
Virtual Function in Derived class

Process returned 0 (0x0)   execution time : 0.141 s
```