Course Name: A Level (2nd Sem) **Subject:** Data Struture using C++
Topic: Pure Virtual Function **Date:** 31-03-2020

Pure Virtual Function

Sometimes implementation of all function cannot be provided in a base class because we don't know the implementation. Such a class is called abstract class. For example, let Shape be a base class. We cannot provide implementation of function draw() in Shape, but we know every derived class must have implementation of draw(). Similarly an Animal class doesn't have implementation of move() (assuming that all animals move), but all animals must know how to move. We cannot create objects of abstract classes.

A pure virtual function (or abstract function) in C++ is a virtual function for which we don't have implementation, we only declare it. A pure virtual function is declared by assigning 0 in declaration. See the following example.

```
// An abstract class
class Test
{
   // Data members of class
public:
   // Pure Virtual Function
   virtual void show() = 0;

  /* Other members */
};
```

A complete example:
A pure virtual function is implemented by classes which are derived from a Abstract class. Following is a simple example to demonstrate the same.

```
#include<iostream>
using namespace std;
class Base
{
  int x;
public:
   virtual void fun() = 0;
   int getX() { return x; }
};

// This class inherits from Base and implements fun()
class Derived: public Base
{
   int y;
public:
   void fun() { cout << "fun() called"; }
};

int main(void)
{
   Derived d;
   d.fun();
   return 0;
```

```
}
```
Output:
```
fun() called
```

Some Interesting Facts:
1) *A class is abstract if it has at least one pure virtual function.*
In the following example, Test is an abstract class because it has a pure virtual function show().

```
// pure virtual functions make a class abstract
#include<iostream>
using namespace std;
 class Test
{
   int x;
public:
    virtual void show() = 0;
    int getX() { return x; }
};

int main(void)
{
    Test t;
    return 0;
}
```

Output:
```
Compiler Error: cannot declare variable 't' to be of abstract  type 'Test' because the following virtual
functions are pure within 'Test': note:      virtual void Test::show()
```

2) *We can have pointers and references of abstract class type.*
For example the following program works fine.
```
#include<iostream>
using namespace std;
 class Base
{
public:
    virtual void show() = 0;
};

class Derived: public Base
{
public:
    void show() { cout << "In Derived \n"; }
};

int main(void)
{
    Base *bp = new Derived();
    bp->show();
    return 0;
}
```

Output:
```
In Derived
```

3) *If we do not override the pure virtual function in derived class, then derived class also becomes abstract class.*
The following example demonstrates the same.

```
#include<iostream>
using namespace std;
class Base
{
public:
    virtual void show() = 0;
};

class Derived : public Base { };
 int main(void)
{
  Derived d;
  return 0;
}
```

```
Compiler Error: cannot declare variable 'd' to be of abstract type 'Derived'  because the following virtual functions are pure within 'Derived': virtual void Base::show()
```

4) *An abstract class can have constructors.*
For example, the following program compiles and runs fine.

```
#include<iostream>
using namespace std;
 class Base
{
protected:
   int x;
public:
  virtual void fun() = 0;
  Base(int i) { x = i; }
};

class Derived: public Base
{
    int y;
public:
    Derived(int i, int j):Base(i) { y = j; }
    void fun() { cout << "x = " << x << ", y = " << y; }
};

int main(void)
{
    Derived d(4, 5);
    d.fun();
    return 0;
}
```

Output:

```
x = 4, y = 5
```