Tutorial 56 - Virtual Functions in C++

Definition:

- A **virtual function** is a member function in the **base class** declared using the `virtual` keyword.
- Virtual functions allow **runtime polymorphism**, enabling the **derived class function** to override the base class function.

Code Snippet 1: Virtual Function Example

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

class BaseClass {
public:
    int var_base = 1;
    virtual void display() { // Virtual function
        cout << "1 Displaying Base class variable var_base: " << var_base << endl;
    }
};

class DerivedClass : public BaseClass {
public:
    int var_derived = 2;
    void display() override { // Overrides base class display
        cout << "2 Displaying Base class variable var_base: " << var_base << endl;
        cout << "2 Displaying Derived class variable var_derived: " << var_derived << endl;
    }
};

```

Explanation of Code Snippet 1:

1. **BaseClass**:
   - Contains:
     - Public data member `var_base` with value `1`.
     - Virtual function `display` to print the value of `var_base`.
2. **DerivedClass**:
   - Inherits **BaseClass**.
   - Contains:
     - Additional data member `var_derived` with value `2`.
     - Overrides the `display` function to print both `var_base` and `var_derived`.

Code Snippet 2: Main Program

```cpp
int main() {
    BaseClass* base_class_pointer;
    BaseClass obj_base;
    DerivedClass obj_derived;
```

```
5
6       base_class_pointer = &obj_derived; // Base class pointer pointing to derived class object
7       base_class_pointer->display();    // Calls DerivedClass display due to virtual function
8
9       return 0;
10  }
11
```

Explanation of Code Snippet 2:

1. **Base Class Pointer**:
   - Created as `base_class_pointer` of type `BaseClass*`.
   - Points to `obj_derived` (an object of **DerivedClass**).
2. **Function Call Behavior**:
   - `display` function is called using `base_class_pointer`.
   - Due to the `virtual` keyword, the **DerivedClass display** function is executed.
3. **Key Point**:
   - If the `virtual` keyword was not used, the **BaseClass display** function would be called, regardless of the pointer pointing to a derived class object.

Output:

```
1  2 Displaying Base class variable var_base: 1
2  2 Displaying Derived class variable var_derived: 2
3
```

Key Notes:

1. **Virtual Functions**:
   - Declared using the `virtual` keyword in the base class.
   - Enables runtime polymorphism.
   - Allows derived classes to override base class functions.
2. **Function Call Behavior**:
   - If a base class pointer points to a derived class object:
     - Without `virtual`, the **base class function** is called.
     - With `virtual`, the **derived class function** is called.
3. **Advantages**:
   - Provides flexibility and extensibility in object-oriented programming.
   - Supports dynamic method dispatch.

Short Notes for Notebook:

Virtual Functions:

1. Declared using the `virtual` keyword in the base class.
2. Enables **runtime polymorphism** by allowing derived class functions to override base class functions.
3. **Key Behavior**:

- Base class pointer pointing to a derived class object calls the **derived class function** if the base class function is virtual.

4. Use the `override` specifier in the derived class for clarity.
5. **Syntax**:

```
virtual void functionName();

```

Program Behavior:

1. Without `virtual`:
   - Base class function is called.
2. With `virtual`:
   - Derived class function is called if the pointer points to a derived object.

Output Example:

```
Derived class function displays both base and derived variables.
```