

Day 8

Friend Function, Pure Virtual, FileHandling, Templates

C++ Inheritance

C++ friend Function and friend Classes

Data hiding is a fundamental concept of object-oriented programming. It restricts the access of private members from outside of the class.

Similarly, protected members can only be accessed by derived classes and are inaccessible from outside.

This video is sole property of Talent Battle Pvt. Ltd. Strict penal action will be taken against unauthorized piracy of this video

C++ Inheritance

friend Function in C++

A **friend function** can access the private and protected data of a class. We declare a friend function using the friend keyword inside the body of the class.

```
class className {  
    ... ..  
    friend returnType functionName(arguments);  
    ... ..  
}
```

This video is sole property of Talent Battle Pvt. Ltd. Strict penal action will be taken against unauthorized piracy of this video

```
// C++ program to demonstrate the working of friend function
```

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

```
class Distance {
private:
    int meter;

    // friend function
    friend int addFive(Distance);

public:
    Distance(): meter(0){}
};
```

```
// friend function definition
int addFive(Distance d){
    // Accessing private members from the friend function
    d.meter += 5;
    return d.meter;
}
```

```
int main(){
    Distance D;
    cout << "Distance: " << addFive(D);
    return 0;
}
```

```
/*
Distance: 5
-----
Process exited after 0.1592 seconds with return value 0
Press any key to continue . . .
*/
```

```
//=====
// Add members of two different classes using friend functions
```

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

```
// forward declaration
class ClassB;
```

```

class ClassA{
public:
    // constructor to initialize numA to 12
    ClassA(): numA(12){}

private:
    int numA;
    // friend function declaration
    friend int add(ClassA, ClassB);
};

class ClassB{
public:
    // constructor to initialize numB to 1
    ClassB(): numB(1){}

private:
    int numB;
    // friend function declaration
    friend int add(ClassA, ClassB);
};

// access members of both classes
int add(ClassA objectA, ClassB objectB){
    return (objectA.numA + objectB.numB);
}

int main(){
    ClassA objectA;
    ClassB objectB;
    cout << "Sum: " << add(objectA, objectB);
    return 0;
}

/*
Sum: 13
-----
Process exited after 0.1271 seconds with return value 0
Press any key to continue . . .
*/

```

C++ Inheritance

friend Class in C++

We can also use a friend Class in C++ using the friend keyword. For example,

```
class ClassB;
class ClassA {
    // ClassB is a friend class of ClassA
    friend class ClassB;
    ... ..
}
class ClassB {
    ... ..
}
```

When a class is declared a friend class, all the member functions of the friend class become friend functions.

Since ClassB is a friend class, we can access all members of ClassA from inside ClassB.

However, we cannot access members of ClassB from inside ClassA. It is because friend relation in C++ is only granted, not taken.

This video is sole property of Talent Battle Pvt. Ltd. Strict penal action will be taken against unauthorized piracy of this video

Slide 246 of 299

```
//=====
// C++ program to demonstrate the working of friend class
```

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

// forward declaration
class ClassB;

class ClassA{
private:
    int numA;

    // friend class declaration
    friend class ClassB;

public:
    // constructor to initialize numA to 12
    ClassA(): numA(12){}
};

class ClassB{
private:
    int numB;

public:
    // constructor to initialize numB to 1
    ClassB(): numB(1){}
```

```

// member friend to add numA
// from ClassA and numB from ClassB
int add(){
    ClassA objectA;
    return objectA.numA + numB;
}
};

int main(){
    ClassB objectB;
    cout << "Sum: " << objectB.add();
    return 0;
}

/*
Sum: 13
-----
Process exited after 0.1059 seconds with return value 0
Press any key to continue . . .
*/

```

PowerPoint Slide Show - [C++ Technical Training CS] - PowerPoint

C++ Inheritance

C++ Virtual Functions

A virtual function is a member function in the base class that we expect to redefine in derived classes.

Basically, a virtual function is used in the base class in order to ensure that the function is **overridden**. This especially applies to cases where a pointer of base class points to an object of a derived class.

This video is sole property of Talent Battle Pvt. Ltd. Strict penal action will be taken against unauthorized piracy of this video

Slide 248 of 299

```

//=====

// C++ program to demonstrate the use of virtual functions

#include<iostream>
using namespace std;

class Base{
public:
    virtual void print(){
        cout << "Base Function" << endl;
    }
};

class Derived : public Base {
public:
    void print(){
        cout << "Derived function" << endl;
    }
};

int main(){
    Derived derived1;

    // pointer of Base type that points to derived1
    Base*base1 = &derived1;

    // class member function of Derived class
    base1 -> print();
    return 0;
}

// Virtual functions are runtime polymorphism.

/*
Derived function

-----
Process exited after 0.1403 seconds with return value 0
Press any key to continue . . .
*/

```

C++ Inheritance

```
class Base {
public:
    virtual void print() {
        // code
    }
};

class Derived : public Base {
public:
    void print() {
        // code
    }
};

int main() {
    Derived derived1;
    Base* base1 = &derived1;

    base1->print();

    return 0;
}
```

print() of Derived
class is called
because print()
of Base class is
virtual

This video is sole property of Talent Battle Pvt. Ltd. Strict penal action will be taken against unauthorized piracy of this video

Slide 250 of 299

C++ Inheritance

Abstract Class and Pure Virtual Function in C++

Abstract Class is a class which contains at least one Pure Virtual function in it.

Abstract classes are used to provide an Interface for its sub classes.

Classes inheriting an Abstract Class must provide definition to the pure virtual function, otherwise they will also become abstract class.

This video is sole property of Talent Battle Pvt. Ltd. Strict penal action will be taken against unauthorized piracy of this video

Slide 251 of 299

C++ Inheritance**Pure Virtual Functions in C++**

Pure virtual Functions are virtual functions with no definition.

They start with virtual keyword and ends with = 0.

Here is the syntax for a pure virtual function,

```
virtual void f() = 0;
```

This video is sole property of Talent Battle Pvt. Ltd. Strict penal action will be taken against unauthorized piracy of this video

Slide 252 of 299

C++ Inheritance**Pure Virtual Functions in C++**

Pure virtual Functions are virtual functions with no definition.

They start with virtual keyword and ends with = 0.

Here is the syntax for a pure virtual function,

```
virtual void f() = 0;
```

```
virtual void print()=0;
```

```
virtual void print()  
{  
    cout << "Hello";  
}
```

This video is sole property of Talent Battle Pvt. Ltd. Strict penal action will be taken against unauthorized piracy of this video

Slide 252 of 299

C++ Inheritance

```
class Base
{
public:
    virtual void show() = 0; // Pure Virtual Function
};
class Derived:public Base
{
public:
    void show()
    {
        cout << "Implementation of Virtual Function in Derived class\n";
    }
};
int main()
{
    Base obj; //Compile Time Error
    Base *b;
    Derived d;
    b = &d;
    b->show();
}
```

This video is sole property of Talent Battle Pvt. Ltd. Strict penal action will be taken against unauthorized piracy of this video

C++ Inheritance

```
class Base
{
public:
    virtual void show() = 0; // Pure Virtual Function
};
class Derived:public Base
{
public:
    void show()
    {
        cout << "Implementation of Virtual Function in Derived class\n";
    }
};
int main()
{
    Base obj; //Compile Time Error
    Base *b;
    Derived d;
    b = &d;
    b->show();
}
```

→ Abstract class

virtual void add()=0;

virtual void add()

}
==

cannot create the object
of the abstract class

→ object of
derived class

This video is sole property of Talent Battle Pvt. Ltd. Strict penal action will be taken against unauthorized piracy of this video

```
//=====
// pure virtual function
```

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

```
class Base{
public:
    virtual void show()=0; // pure virtual function
};

class Derived: public Base{
public:
    void show(){
        cout << "Implementation of Virtual funciton in Derived class\n";
    }
};

int main(){
    //Base obj; // Compile Time Error
    Base *b;
    Derived d;
    b = &d;
    b ->show();
    return 0;
}

/*
Implementation of Virtual funciton in Derived class

-----
Process exited after 0.12 seconds with return value 0
Press any key to continue . . .

*/
```

PowerPoint Slide Show - [C++ Technical Training CS] - PowerPoint

C++ Inheritance

File Handling using File Streams in C++

File represents storage medium for storing data or information. Streams refer to sequence of bytes. In Files we store data i.e. text or binary data permanently and use these data to read or write in the form of input output operations by transferring bytes of data. So we use the term File Streams/File handling. We use the header file **<fstream>**

ofstream: It represents output Stream and this is used for writing in files.
ifstream: It represents input Stream and this is used for reading from files.
fstream: It represents both output Stream and input Stream. So it can read from files and write to files.

Operations in File Handling:

Creating a file: `open()`
Reading data: `read()`
Writing new data: `write()`
Closing a file: `close()`

Slide 254 of 299

This video is sole property of Talent Battle Pvt. Ltd. Strict penal action will be taken against unauthorized piracy of this video

PowerPoint Slide Show - [C++ Technical Training CS] - PowerPoint

C++ Inheritance

We create/open a file by specifying new path of the file and mode of operation.

Operations can be reading, writing, appending and truncating.

Syntax for file creation: **`FilePointer.open("Path",ios::mode);`**

Example of file opened for writing:
`st.open("E:\program.txt",ios::out);`

Example of file opened for reading:
`st.open("E:\program.txt",ios::in);`

Example of file opened for appending:
`st.open("E:\program.txt",ios::app);`

Example of file opened for truncating:
`st.open("E:\program.txt",ios::trunc);`

Slide 255 of 299

This video is sole property of Talent Battle Pvt. Ltd. Strict penal action will be taken against unauthorized piracy of this video

```
//=====
// file handling

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

```

int main(){
    fstream st; // step 1: creating object of fstream class
    st.open("C:/Users/hp/Desktop/TCS IT/C++ language/Day8/program.txt", ios::out); // Step 2:
create new file
    if(!st) // step 3: checking whether file exist
    {
        cout << "File creation failed";
    } else {
        cout << "New file created";
        st.close(); // step 4: Closing file
    }

    return 0;
}

/*
New file created
-----
Process exited after 0.1531 seconds with return value 0
Press any key to continue . . .
*/

//=====
// file handling --> writing too a file

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

int main(){
    fstream st; // Step 1: create object of fstream class
    st.open("C:/Users/hp/Desktop/TCS IT/C++ language/Day8/program.txt", ios::out); // Step 2:
creating new file
    if(!st) // step 3: checking whether file exist
    {
        cout << "File creation failed";
    } else {
        cout << "New file created";
        st << "Hello"; // step 4: writing to file
        st.close(); // step 5: closing file
    }
    return 0;
}

/*
New file created

```

```
-----
Process exited after 0.01468 seconds with return value 0
Press any key to continue . . .
*/
```

PowerPoint Slide Show - [C++ Technical Training CS] - PowerPoint

C++ Inheritance

Special operations in a File

There are few important functions to be used with file streams like:

tellp() - It tells the current position of the put pointer.
Syntax: `filepointer.tellp()`

tellg() - It tells the current position of the get pointer.
Syntax: `filepointer.tellg()`

seekp() - It moves the put pointer to mentioned location.
Syntax: `filepointer.seekp(no of bytes,reference mode)`

seekg() - It moves get pointer(input) to a specified location.
Syntax: `filepointer.seekg((no of bytes,reference point)`

put() - It writes a single character to file.
get() - It reads a single character from file.
Note: For `seekp` and `seekg` three reference points are passed: `ios::beg` - beginning of the file `ios::cur` - current position in the file `ios::end` - end of the file

This video is sole property of Talent Battle Pvt. Ltd. Strict penal action will be taken against unauthorized piracy of this video

Slide 260 of 299

34:48:15 1.5x

```
//=====
// file handling --> Reading from a file

/*
#include<iostream>
#include<fstream>
using namespace std;

int main(){
    fstream st; // Step 1: create object of fstream class
    st.open("C:/Users/hp/Desktop/TCS IT/C++ language/Day8/program.txt", ios::in); // Step 2:
creating new file
    if(!st) // step 3: checking whether file exist
    {
        cout << "No such file";
    } else{
        char ch;
        while(!st.eof()){
            st >> ch; // step 4: Reading from file
            cout << ch; // Message Read from file
        }
    }
}
```

```

    st.close(); // step 5 : closing file
}
return 0;
}
*/

/*
This is C++ file
-----
Process exited after 0.01591 seconds with return value 0
Press any key to continue . . .
*/

//=====
// close a file
// It is done by FilePointer.close()

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

int main(){
    fstream st; // step 1: creating object of fstream class
    st.open("C:/Users/hp/Desktop/TCS IT/C++ language/Day8/program.txt", ios::out); // step 2:
creating new file
    st.close(); // step 3 : closing file

    return 0;
}

/*
-----
Process exited after 0.01027 seconds with return value 0
Press any key to continue . . .
*/

```

```

//=====

```

```

#include<iostream>
#include<fstream>
using namespace std;
int main(){
    fstream st; // creating object of fstream class
st.open("C:/Users/hp/Desktop/TCS IT/C++ language/Day8/program.txt", ios::out); // create new
file
    if(!st) // checking whether file exist
    {
        cout << "File creation failed";
    } else {
        cout << "New file created" << endl;
        st << "Hello Friends"; // writing to file

        // checking the file pointer position
        cout << "File pointer position is " << st.tellp() << endl;

        st.seekp(-1, ios::cur); // Go one position back from current position

        // checking the file pointer position
        cout << "As per tellp file pointer position is " << st.tellp() << endl;

        st.close(); // closing file
    }

    st.open("C:/Users/hp/Desktop/TCS IT/C++ language/Day8/program.txt", ios::in); // opening
file in read mode
    if(!st) // checking whether file exist
    {
        cout << "No such file";
    } else {
        char ch;
        st.seekg(5, ios::beg); // go to position 5 from beging.
        cout << "As per tellg File pointer position is " << st.tellg() << endl; // checking file
pointer position
        cout << endl;
        st.seekg(1, ios::cur); // Go to position 1 from beginning
        st.close(); // closing file
    }
    return 0;
}

/*
New file created
File pointer position is 13
As per tellp file pointer position is 12
As per tellg File pointer position is 5
-----

```

Process exited after 0.1185 seconds with return value 0

Press any key to continue . . .

*/

C++ Inheritance**C++ Templates**

Templates are powerful features of C++ which allows you to write generic programs. In simple terms, you can create a single function or a class to work with different data types using templates.

Templates are often used in larger codebase for the purpose of code reusability and flexibility of the programs.

The concept of templates can be used in two different ways:

- Function Templates
- Class Templates

This video is sole property of Talent Battle Pvt. Ltd. Strict penal action will be taken against unauthorized piracy of this video

Slide 262 of 299

C++ Inheritance**Function Templates**

A function template works in a similar to a normal **function**, with one key difference.

A single function template can work with different data types at once but, a single normal function can only work with one set of data types.

Normally, if you need to perform identical operations on two or more types of data, you use function overloading to create two functions with the required function declaration.

However, a better approach would be to use function templates because you can perform the same task writing less and maintainable code.

This video is sole property of Talent Battle Pvt. Ltd. Strict penal action will be taken against unauthorized piracy of this video

Slide 263 of 299

C++ Inheritance

How to declare a function template?

A function template starts with the keyword `template` followed by template parameter/s inside `< >` which is followed by function declaration.

```
template <class T>
T someFunction(T arg)
{
    ... ..
}
```

In the above code, T is a template argument that accepts different data types (int, float), and `class` is a keyword.

You can also use keyword `typename` instead of `class` in the above example.

When, an argument of a data type is passed to `someFunction()`, compiler generates a new version of `someFunction()` for the given data type.

This video is sole property of Talent Battle Pvt. Ltd. Strict penal action will be taken against unauthorized piracy of this video

Slide 264 of 299

```
//=====
// Program to display largest among two numbers using function templates.
// If two characters are passed to function template, character with larger ASCII value is
// displayed

#include <iostream>
using namespace std;

// template function
template <class T>
T Large(T n1, T n2){
    return (n1 > n2) ? n1 : n2;
}

int main(){
    int i1, i2;
    float f1, f2;
    char c1, c2;

    cout << "Enter two integers:\n";
    cin >> i1 >> i2;
    cout << Large(i1, i2) << " is larger. " << endl;

    cout << "\n Enter two floating-point numbers:\n";
    cin >> f1 >> f2;
    cout << Large(f1, f2) << "is larger." << endl;
```

```
cout << "\nEnter two characters:\n";
cin >> c1 >> c2;
cout << Large(c1, c2) << " has larger ASCII value.";

return 0;
}
```

```
/*
```

```
Enter two integers:
```

```
55
```

```
66
```

```
66 is larger.
```

```
Enter two floating-point numbers:
```

```
33
```

```
22
```

```
33 is larger.
```

```
Enter two characters:
```

```
h
```

```
l
```

```
l has larger ASCII value.
```

```
-----
```

```
Process exited after 18.82 seconds with return value 0
```

```
Press any key to continue . . .
```

```
*/
```