## Q no: 1 Friend Function

A friend function in C++ is a function that is not a member of a class but has access to the class's private and protected members. It is declared inside the class with the keyword **"friend"** preceding the function prototype.

**How does a friend function differ from a member function?**

|  |  |
| --- | --- |
| **Friend Function** | **Member Function** |
| A friend function is not part of the class but has been granted access to its private and protected members. | A member function is part of the class and has access to the class's private and protected members directly |

**Purpose of using friend functions:**

The purpose of using friend functions in a C++ class is to allow specific non-member functions to access the private and protected members of the class without having to make those members public. This can be useful in situations where you want to provide access to certain functions without exposing the internal details of the class to the outside world. It can also be used to achieve operator overloading, where non-member functions are given access to private members of a class to perform operations.

## Q no: 2

**How does a friend function bypass the access restrictions of a class? Explain with**

**suitable example.**

A friend function in C++ bypasses the access restrictions of a class by being explicitly granted access to the private and protected members of the class. This means that a friend function can directly access and modify the private and protected members of the class, even though it is not a member of the class itself.

**Example:**

#include <iostream>

using namespace std;

class MyClass {

private:

int privateMember;

public:

MyClass() : privateMember(0) {}

void setPrivateMember(int value) {

privateMember = value;

}

};

void friendFunction(MyClass obj) {

cout << "Friend function accessing private member: " << obj.privateMember << endl;

}

int main() {

MyClass myObject;

myObject.setPrivateMember(42);

friendFunction(myObject);

return 0;

} **Output:**

****

## Q no: 3 this keyword

In object-oriented programming, the "this" keyword refers to the current object instance within a class. It is a pointer that holds the memory address of the current object. The "this" pointer is implicitly available within non-static member functions of a class and can be used to access members of the class or to pass the current object as a parameter to other functions.

**Explain how the ‘this’ pointer is used to differentiate between class members and parameters with the same name**.

The "this" pointer is particularly useful when differentiating between class members and parameters with the same name. Consider the following example:

#include <iostream>

using namespace std;

class MyClass {

private:

int value;

public:

void setValue(int value) {

this->value = value; // Using "this" to differentiate between class member and parameter

}

int getValue() {

return this->value; // Using "this" to access the class member

}

};

int main() {

MyClass obj;

obj.setValue(10);

cout << obj.getValue() << endl;

return 0;

}

**Output**



In this example, the "this" pointer is used to differentiate between the class member `value` and the parameter `value` in the `setValue` method. When we write `this->value = value;`, we are explicitly referring to the class member `value` and assigning it the value of the parameter `value`. Similarly, in the `getValue` method, `this->value` is used to access the class member `value` and return its value.

By using the "this" pointer, we can avoid naming conflicts and explicitly specify which variable we are referring to within the class, making the code more readable and less error-prone.

## Q no: 4

**How does the “this” pointer work when used as a pointer to an object? Provide an**

**example demonstrating the use of the “this” pointer in a member function.**

When used as a pointer to an object, the "this" pointer in C++ holds the memory address of the current object instance. It allows member functions to access the members of the object they are called on. Here's an example demonstrating the use of the "this" pointer in a member function:

#include <iostream>

using namespace std;

class MyClass {

private:

int value;

public:

MyClass(int value) : value(value) {}

void setValue(int value) {

this->value = value; // Using "this" to access the member variable

}

void printAddress() {

cout << "Memory address of the current object: " << this << endl;

}

};

int main() {

MyClass obj1(30);

MyClass obj2(20);

obj1.printAddress();

obj2.printAddress();

return 0;

}

**Output**



In this example, the "this" pointer is used in the `printAddress` method to print the memory address of the current object. When `obj1.printAddress()` and `obj2.printAddress()` are called, the "this" pointer holds the memory address of `obj1` and `obj2` respectively, allowing the member function to access the specific object's address.

The "this" pointer is implicitly available within non-static member functions of a class and can be used to access members of the object it is called on. It is particularly useful when you need to refer to the current object within its member functions.

## Q no: 5

**Can the “this” pointer be used in static member functions? Why or why not? Discuss the role of the “this” pointer in the context of static vs. non-static member functions.**

No, the "this" pointer cannot be used in static member functions in C++. The "this" pointer is a special pointer that holds the memory address of the current object instance. However, static member functions do not have a "this" pointer because they are not associated with any particular object instance. They are class-level functions and do not have access to any specific object's data.

|  |  |
| --- | --- |
| **Non-static member functions** | **Static member functions** |
| Non-static member functions have access to the "this" pointer, which allows them to access the members of the object they are called on. They operate on a specific object's data and can modify the state of the object. | Static member functions do not have access to the "this" pointer because they are not tied to any specific object. They are associated with the class itself rather than with any particular object instance. As a result, they cannot access non-static members of the class directly using the "this" pointer. |

In summary, the "this" pointer is a fundamental concept in C++ that allows non-static member functions to access the members of the object they are called on. However, it is not available in static member functions, which operate at the class level and do not have access to specific object instances.

## Q no: 6 File Handling

In C++, file handling is implemented through the use of the `<fstream>` library, which provides classes and functions for working with files. The key components of file handling in C++ include the following:

1. **File Streams:** C++ provides three classes for working with files: `ifstream` (input file stream) for reading from files, `ofstream` (output file stream) for writing to files, and `fstream` (file stream) for both reading and writing.

2. **File Opening:** Files are opened using the `open()` method of file stream objects, specifying the file name and the mode (e.g., input, output, append) as parameters.

3. **File Reading and Writing**: Once a file is opened, data can be read from or written to the file using standard input/output operations such as `<<` and `>>` for formatted input/output, and `write()` and `read()` for binary input/output.

**Differentiating between text and binary files in the context of C++ file handling:**

|  |  |
| --- | --- |
| **Text files** | **Binary files** |
| Text files are human-readable files that store data in a format that can be understood by a human.  When working with text files in C++, data is typically stored and read as characters and strings.  Text files are suitable for storing textual data such as configuration files, logs, and simple data records. | Binary files store data in a binary format, which means that the data is stored in the same format as it is held in memory. When working with binary files in C++, data is stored and read in its raw binary form.  Binary files are suitable for storing complex data structures, such as objects, arrays, and non-textual data like images, audio, and video |

In summary, C++ file handling involves using file stream classes to open, read from, and write to files. Text files store data in a human-readable format, while binary files store data in a raw binary format. The choice between text and binary files depends on the nature of the data being stored and the requirements of the application.

## Q no: 7

**Briefly describe ofstream, ifstream, fstream with suitbale example using c++.**

In C++, the `<fstream>` library provides three classes for file handling: `ofstream` (output file stream), `ifstream` (input file stream), and `fstream` (file stream). Here's a brief description of each class along with a suitable example:

**1. ofstream (Output File Stream):** This class is used to create and write to files. It is used when you want to write data to a file.

**Example:**

#include <iostream>

#include <fstream>

using namespace std;

int main() {

ofstream outputFile("output.txt"); // Create an output file stream

if (outputFile.is\_open()) {

outputFile << "Hello, this is a sample output to a file." << endl;

outputFile.close(); // Close the file

cout << "Data has been written to the file." << endl;

} else {

cout << "Unable to open the file." << endl;

}

return 0;

} **Output**



**2. ifstream (Input File Stream):** This class is used to read from files. It is used when you want to read data from a file.

**Example:**

#include <iostream>

#include <fstream>

#include <string>

using namespace std;

int main() {

ifstream inputFile("input.txt"); // Create an input file stream

string line;

if (inputFile.is\_open()) {

while (getline(inputFile, line)) {

cout << line << endl; // Output the content of the file

}

inputFile.close(); // Close the file

} else {

cout << "Unable to open the file." << endl;

}

return 0;

} **Output**



**3. fstream (File Stream):** This class is used for both reading from and writing to files. It provides the functionality of both `ofstream` and `ifstream`.

**Example:**

#include <iostream>

#include <fstream>

using namespace std;

int main() {

fstream file("data.txt", ios::out | ios::in); // Create a file stream for both input and output

if (file.is\_open()) {

file << "This is a sample text." << endl; // Write to the file

file.seekg(0); // Move the file pointer to the beginning

string content;

getline(file, content); // Read from the file

cout << "Content of the file: " << content << endl;

file.close(); // Close the file

} else {

cout << "Unable to open the file." << endl;

}

return 0;

} **Output**

****