Here are the C++ programs based on the topics you've provided:

1. **Evaluate the expression y = 5*x - 5**

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

int main() {
    int x, y;
    cout << "Enter the value of x: ";
    cin >> x;
    y = 5 * x - 5;
    cout << "The value of y is: " << y << endl;
    return 0;
}
```

2. **Friend Function and Friend Class**

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

class B; // Forward declaration

class A {
private:
    int data;
public:
    A(int value) : data(value) {}
    friend int add(A objA, B objB);
};

class B {
private:
    int data;
public:
    B(int value) : data(value) {}
    friend int add(A objA, B objB);
};

int add(A objA, B objB) {
    return objA.data + objB.data;
}

int main() {
    A a(10);
    B b(20);
    cout << "Sum: " << add(a, b) << endl;
    return 0;
}
```

3. **Relational Operator**

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

int main() {
    int value;
    cout << "Enter a value: ";
    cin >> value;
    if (value > 10) {
        cout << "hi" << endl;
    } else {
        cout << "bye" << endl;
    }
    return 0;
}
```

4. **Inline Function, Static Data Members & Member Functions**

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

class MyClass {
public:
    static int count;
    MyClass() {
        count++;
    }
    static void displayCount() {
        cout << "Count: " << count << endl;
    }
    inline int multiplyByTwo(int x) {
        return x * 2;
    }
};

int MyClass::count = 0;

int main() {
    MyClass obj1, obj2, obj3;
    MyClass::displayCount();
    cout << "Result: " << obj1.multiplyByTwo(5) << endl;
    return 0;
}
```

5. **Matrix Multiplication**

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

class Matrix {
private:
    int rows, cols;
    int** data;
public:
    Matrix(int r, int c) : rows(r), cols(c) {
        data = new int*[rows];
        for (int i = 0; i < rows; i++) {
            data[i] = new int[cols];
        }
    }
    void input() {
        cout << "Enter matrix elements: " << endl;
        for (int i = 0; i < rows; i++) {
            for (int j = 0; j < cols; j++) {
                cin >> data[i][j];
            }
        }
    }
    Matrix multiply(const Matrix& other) {
        Matrix result(rows, other.cols);
        for (int i = 0; i < rows; i++) {
            for (int j = 0; j < other.cols; j++) {
                result.data[i][j] = 0;
                for (int k = 0; k < cols; k++) {
                    result.data[i][j] += data[i][k] *
other.data[k][j];
                }
            }
        }
        return result;
    }
    void display() {
        for (int i = 0; i < rows; i++) {
            for (int j = 0; j < cols; j++) {
                cout << data[i][j] << " ";
            }
            cout << endl;
        }
    }
};

int main() {
    Matrix m1(2, 2), m2(2, 2);
    m1.input();
    m2.input();
    Matrix result = m1.multiply(m2);
    cout << "Resultant Matrix: " << endl;
    result.display();
    return 0;
```

```
}
```

6. **Classes and Objects**

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

class Dog {
public:
    string name;
    int age;

    void bark() {
        cout << "Woof!" << endl;
    }
};

int main() {
    Dog myDog;
    myDog.name = "Buddy";
    myDog.age = 3;

    cout << "Dog's name: " << myDog.name << endl;
    cout << "Dog's age: " << myDog.age << endl;
    myDog.bark();

    return 0;
}
```

7. **Constructors and Destructors**

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

class MyClass {
public:
    MyClass() {
        cout << "Constructor called" << endl;
    }
    ~MyClass() {
        cout << "Destructor called" << endl;
    }
};

int main() {
    MyClass obj;
    return 0;
}
```

8. **Array of Objects, Pointer to Object**

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

class Car {
public:
    string model;
    int year;

    void display() {
        cout << "Model: " << model << ", Year: " << year << endl;
    }
};

int main() {
    Car cars[3];
    cars[0].model = "Toyota";
    cars[0].year = 2020;
    cars[1].model = "Honda";
    cars[1].year = 2021;
    cars[2].model = "Ford";
    cars[2].year = 2022;

    Car* ptr = &cars[0];
    for (int i = 0; i < 3; i++) {
        ptr->display();
        ptr++;
    }

    return 0;
}
```

9. **THIS Pointer and Dynamic Allocation Operators**

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

class Box {
public:
    int length;
    int breadth;

    Box(int length, int breadth) : length(length),
breadth(breadth) {}

    Box& resize(int l, int b) {
        this->length = l;
        this->breadth = b;
        return *this;
    }
    void display() {
        cout << "Length: " << length << ", Breadth: " << breadth
<< endl;
    }
};

int main() {
    Box* myBox = new Box(10, 5);
    myBox->display();
    myBox->resize(20, 10)->display();

    delete myBox;
    return 0;
}
```

10. **Function Overloading and Operator Overloading**

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

class Complex {
private:
    int real;
    int imag;
public:
    Complex(int r = 0, int i = 0) : real(r), imag(i) {}

    Complex operator+(const Complex& other) {
        return Complex(real + other.real, imag + other.imag);
    }

    Complex operator-() {
        return Complex(-real, -imag);
    }

    bool operator==(const Complex& other) {
        return (real == other.real && imag == other.imag);
    }

    void display() {
        cout << real << " + " << imag << "i" << endl;
    }
};

int main() {
    Complex c1(5, 3), c2(2, 4);
    Complex c3 = c1 + c2;
    c3.display();
    Complex c4 = -c1;
    c4.display();

    if (c1 == c2) {
        cout << "Equal" << endl;
    } else {
        cout << "Not Equal" << endl;
    }

    return 0;
}
```

11. **Virtual Function, Virtual Class, Early and Late Binding**

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

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

class Derived : public Base {
public:
    void display() override {
        cout << "Derived class" << endl;
    }
};

int main() {
    Base* basePtr;
    Base baseObj;
    Derived derivedObj;

    basePtr = &baseObj;
    basePtr->display();

    basePtr = &derivedObj;
    basePtr->display();

    return 0;
}
```

12. **Generic Function & Classes**

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

```
template <typename T>
T maxVal(T a, T b) {
    return (a > b) ? a : b;
}

template <class T>
class MyTemplateClass {
private:
    T data;
public:
    MyTemplateClass(T value) : data(value) {}
    T getData() { return data; }
};

int main() {
    cout << maxVal(5, 10) << endl;
    cout << maxVal(5.5, 10.5) << endl;

    MyTemplateClass<int> intObj(10);
    cout << intObj.getData() << endl;

    MyTemplateClass<double> doubleObj(10.5);
    cout << doubleObj.getData() << endl;

    return 0;
}
```

13. **Single Level Inheritance, Multilevel Inheritance**

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

// Single Level Inheritance
class Animal {
public:
    string name;
    void eat() {
        cout << "Eating" << endl;
    }
};

class Dog : public Animal {
public:
    void bark() {
        cout << "Woof!" << endl;
    }
};
```

```
// Multilevel Inheritance
class Vehicle {
public:
    string model;
};

class Car : public Vehicle {
public:
    int wheels;
};

class SportsCar : public Car {
public:
    int maxSpeed;
};

int main() {
    Dog myDog;
    myDog.name = "Buddy";
    myDog.eat();
    myDog.bark();

    SportsCar mySportsCar;
    mySportsCar.model = "Ferrari";
    mySportsCar.wheels = 4;
    mySportsCar.maxSpeed = 300;

    cout << "Car Model: " << mySportsCar.model << endl;

    return 0;
}
```

14. **File Read and Write**

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

int main() {
    // Write to file
    ofstream myFile("example.txt");
    if (myFile.is_open()) {
        myFile << "Hello, world!\n";
        myFile << "This is a sample text.\n";
        myFile.close();
    } else {
        cout << "Unable to open file for writing" << endl;
    }
    // Read from file
    string line;
    ifstream readFile("example.txt");
    if (readFile.is_open()) {
        while (getline(readFile, line)) {
            cout << line << endl;
        }
        readFile.close();
    } else {
        cout << "Unable to open file for reading" << endl;
    }

    return 0;
}
```

15. **Prime Number**

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

class Number {
private:
    int num;
public:
    Number(int n) : num(n) {}

    bool isPrime() {
        if (num <= 1) return false;
        for (int i = 2; i * i <= num; i++) {
            if (num % i == 0) return false;
        }
        return true;
    }

    void display() {
        if (isPrime()) {
            cout << num << " is a prime number" << endl;
        } else {
            cout << num << " is not a prime number" << endl;
        }
    }
};

int main() {
    Number n1(11);
    n1.display();

    Number n2(12);
    n2.display();

    return 0;
}
```

16. **Factorial**

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

class Factorial {
private:
    int num;
public:
    Factorial(int n) : num(n) {}

    int calculateFactorial() {
        if (num == 0 || num == 1) return 1;
        int result = 1;
        for (int i = 2; i <= num; i++) {
            result *= i;
        }
        return result;
    }

    void display() {
        cout << "Factorial of " << num << " is " <<
calculateFactorial() << endl;
    }
};

int main() {
    Factorial f1(5);
    f1.display();

    Factorial f2(6);
    f2.display();

    return 0;
}
```

17. **Function Overloading**

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

class Calculator {
public:
    int add(int a, int b) {
        return a + b;
    }

    double add(double a, double b) {
        return a + b;
    }

    int add(int a, int b, int c) {
        return a + b + c;
    }
};

int main() {
    Calculator calc;
    cout << calc.add(1, 2) << endl;
    cout << calc.add(1.5, 2.5) << endl;
    cout << calc.add(1, 2, 3) << endl;

    return 0;
}
```