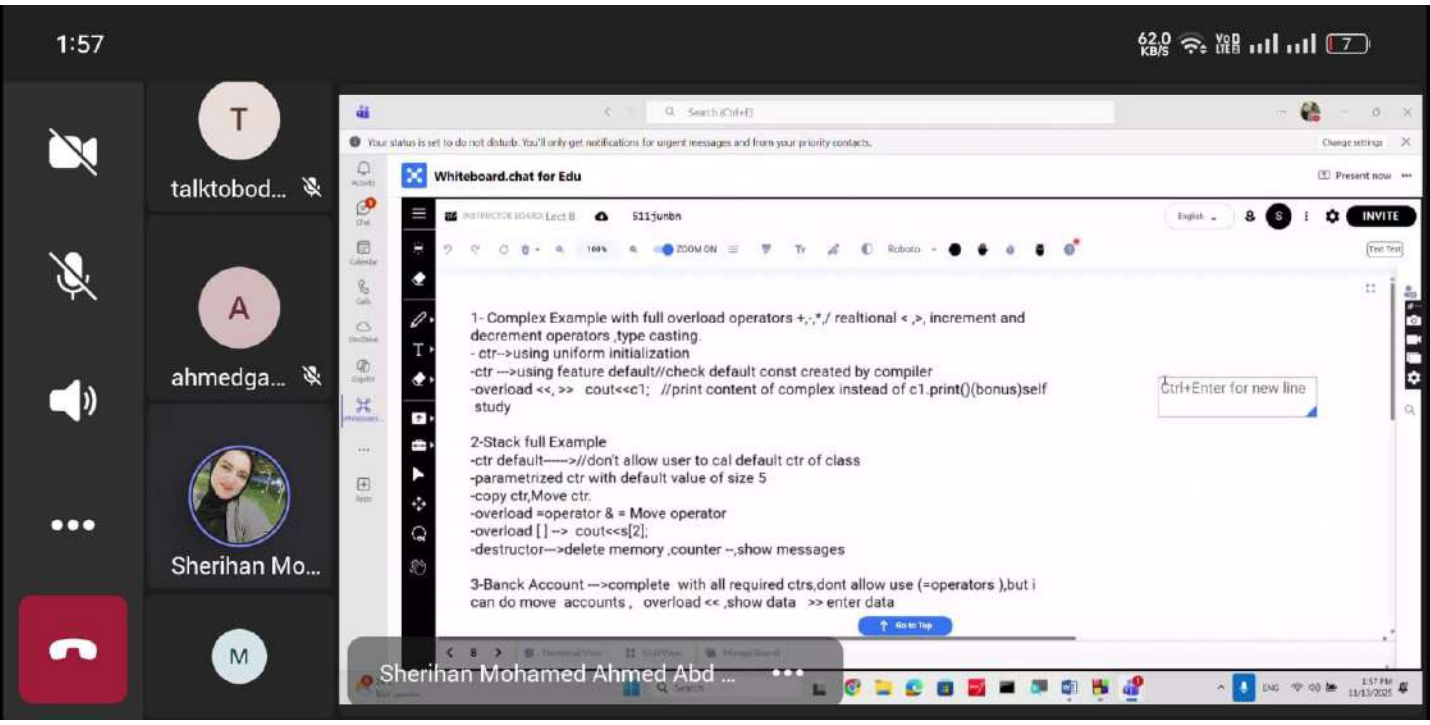
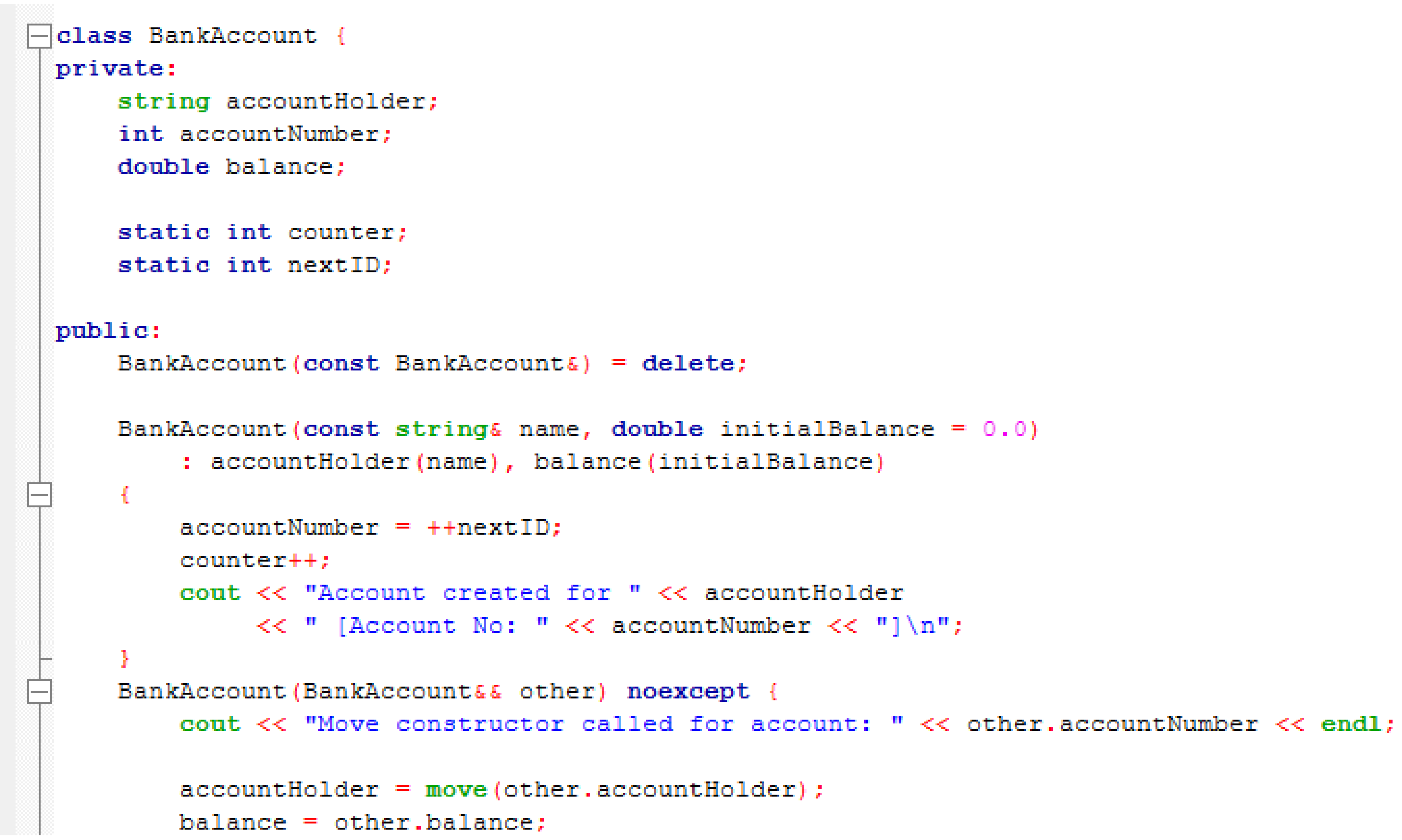


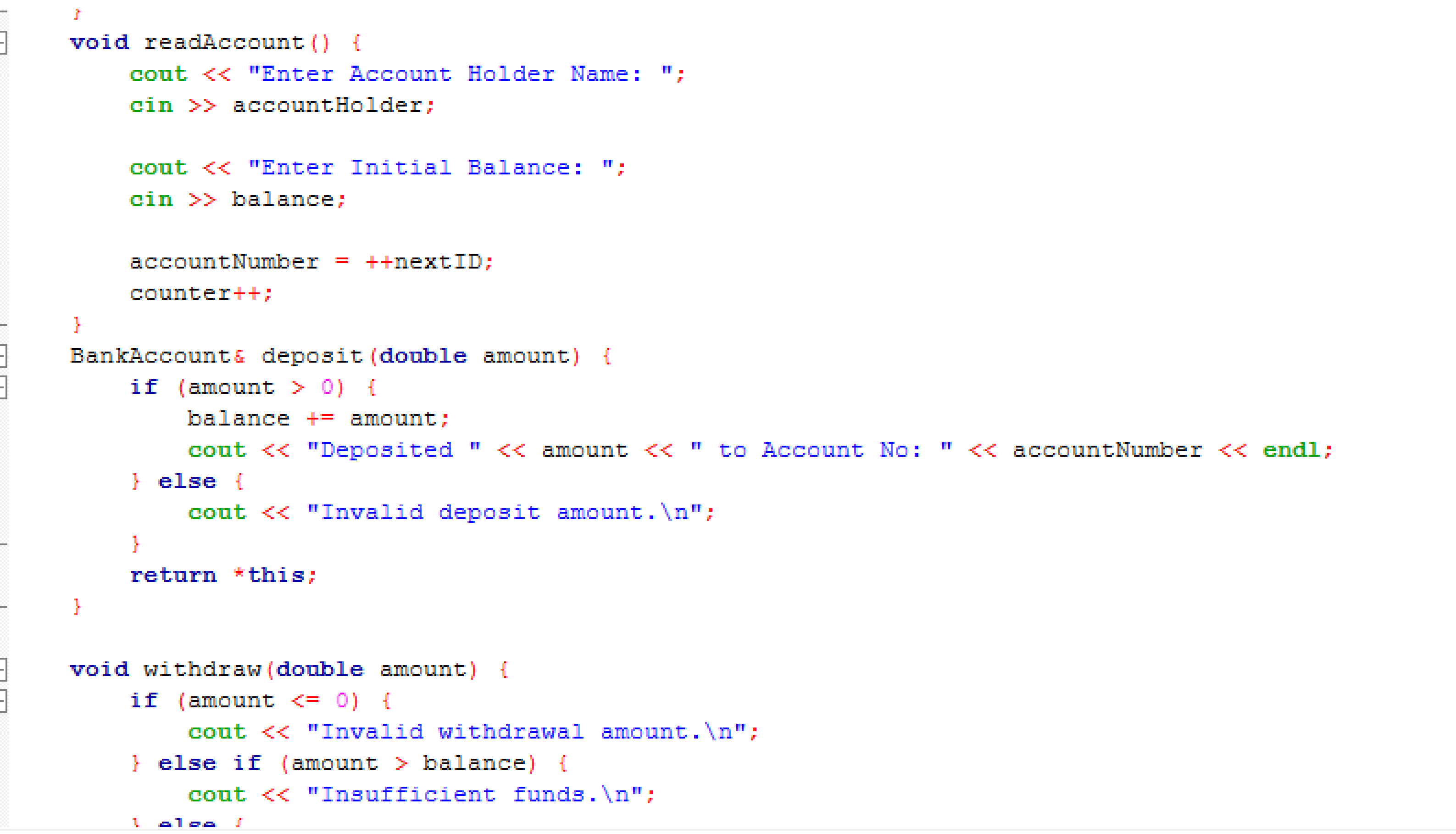
1- Complex Example with full overload operators +,-,*,/ realtional < ,>, increment and decrement operators ,type casting.
- ctr-->using uniform initialization
-ctr --->using feature default//check default const created by compiler
-overload <<, >>   cout<<c1;   //print content of complex instead of c1.print()(bonus)self study

2-Stack full Example
-ctr default------>//don't allow user to cal default ctr of class
-parametrized ctr with default value of size 5
-copy ctr,Move ctr.
-overload =operator & = Move operator
-overload [ ] --> cout<<s[2];
-destructor--->delete memory ,counter --,show messages

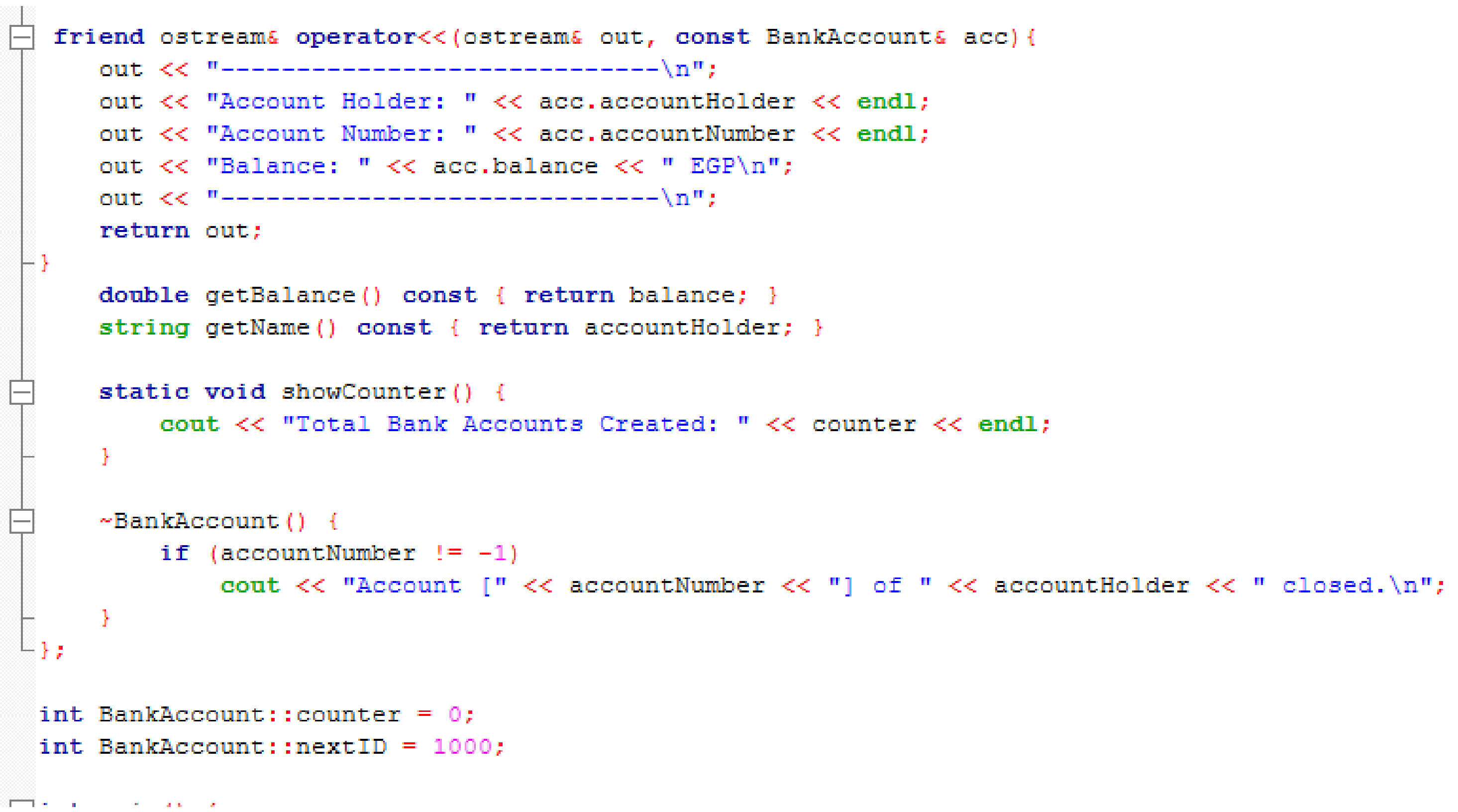
3-Banck Account --->complete with all required ctrs,dont allow use (=operators),but i can do move accounts , overload << ,show data >> enter data



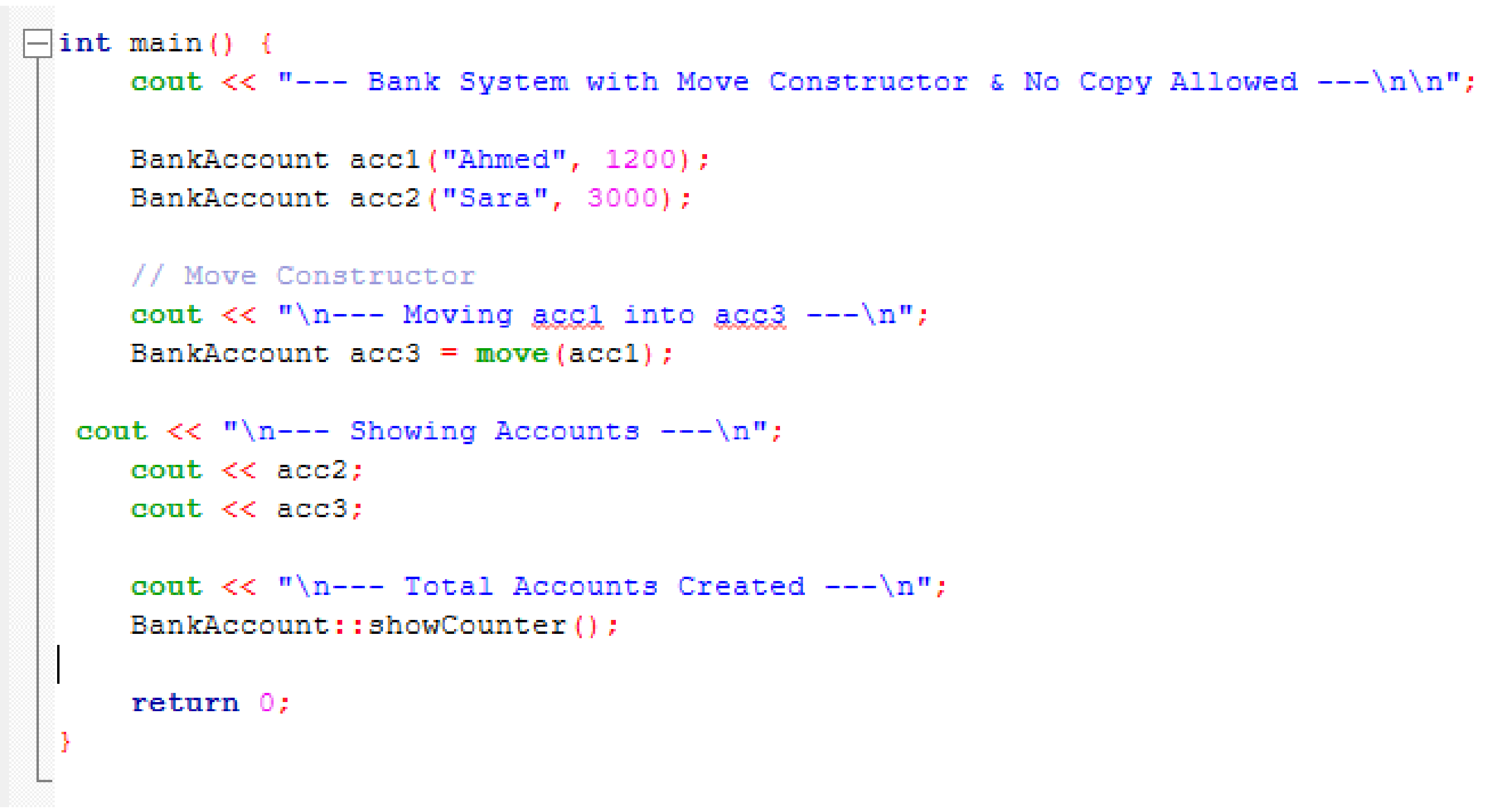
```cpp
class BankAccount {
private:
    string accountHolder;
    int accountNumber;
    double balance;

    static int counter;
    static int nextID;

public:
    BankAccount(const BankAccount&) = delete;

    BankAccount(const string& name, double initialBalance = 0.0)
        : accountHolder(name), balance(initialBalance)
    {
        accountNumber = ++nextID;
        counter++;
        cout << "Account created for " << accountHolder
             << " [Account No: " << accountNumber << "]\n";
    }
    BankAccount(BankAccount&& other) noexcept {
        cout << "Move constructor called for account: " << other.accountNumber << endl;

        accountHolder = move(other.accountHolder);
        balance = other.balance;
```



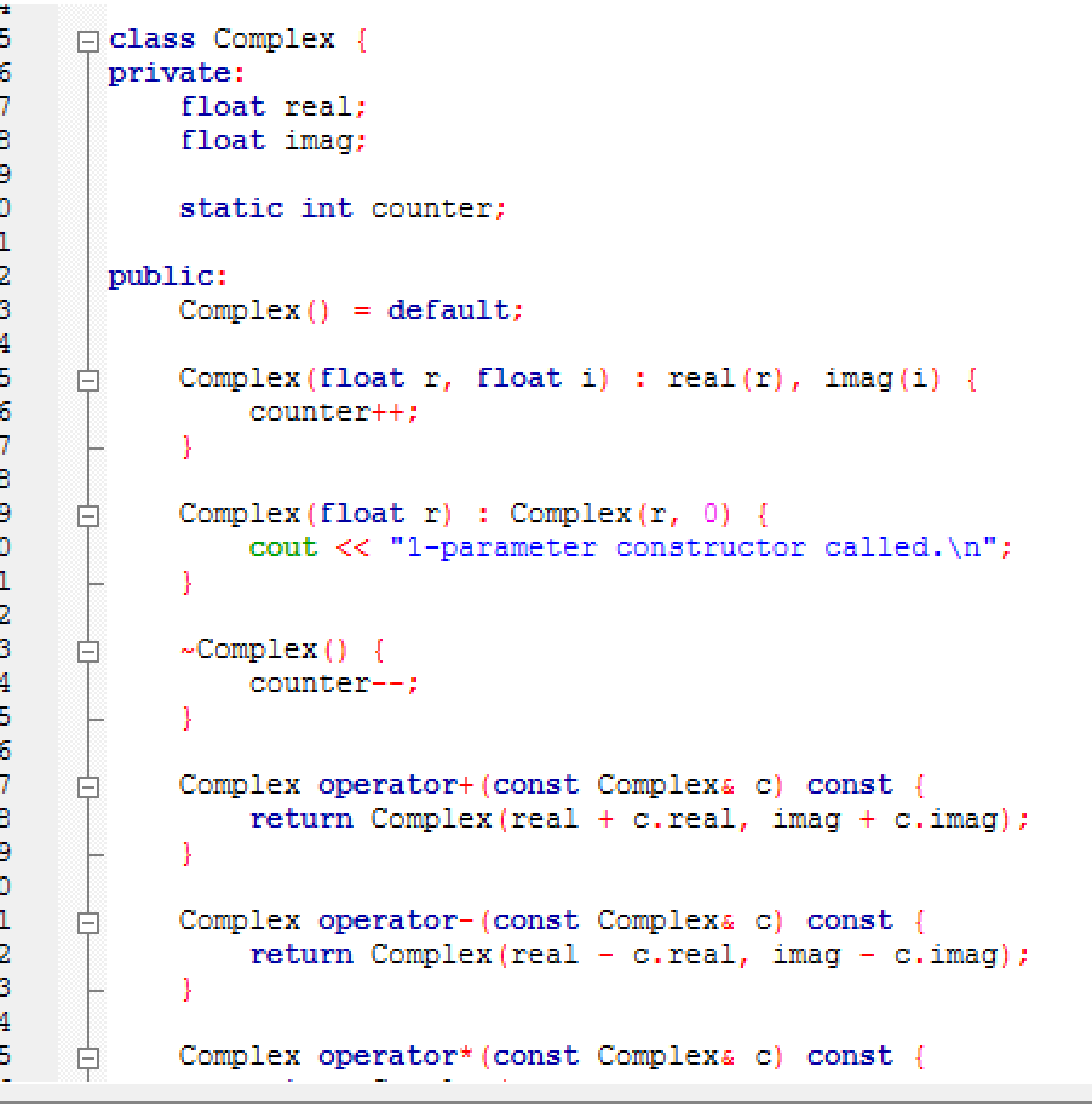
```
}
void readAccount() {
    cout << "Enter Account Holder Name: ";
    cin >> accountHolder;

    cout << "Enter Initial Balance: ";
    cin >> balance;

    accountNumber = ++nextID;
    counter++;
}
BankAccount& deposit(double amount) {
    if (amount > 0) {
        balance += amount;
        cout << "Deposited " << amount << " to Account No: " << accountNumber << endl;
    } else {
        cout << "Invalid deposit amount.\n";
    }
    return *this;
}

void withdraw(double amount) {
    if (amount <= 0) {
        cout << "Invalid withdrawal amount.\n";
    } else if (amount > balance) {
        cout << "Insufficient funds.\n";
    } else {
```



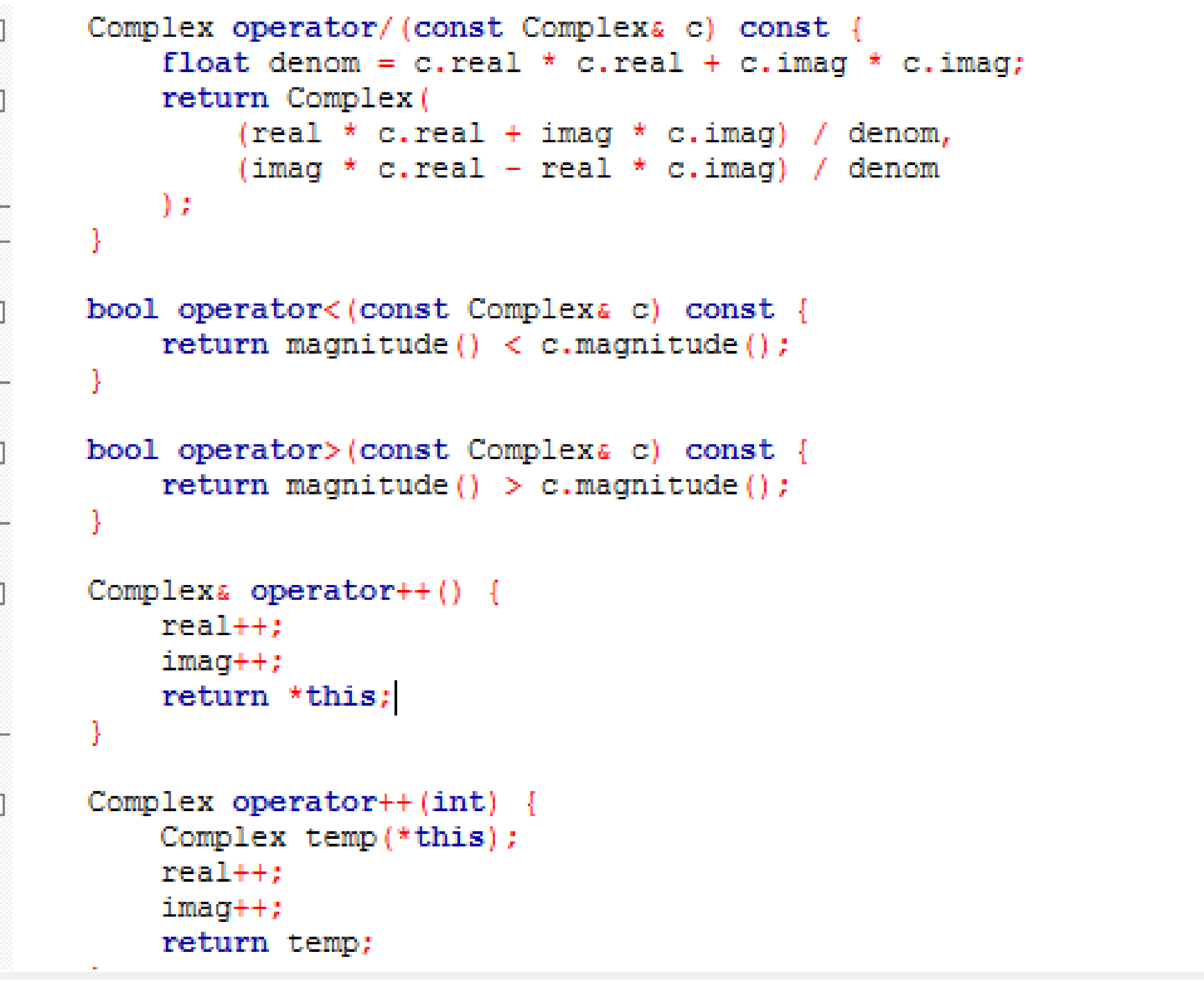
```
 friend ostream& operator<<(ostream& out, const BankAccount& acc){
    out << "------------------------------\n";
    out << "Account Holder: " << acc.accountHolder << endl;
    out << "Account Number: " << acc.accountNumber << endl;
    out << "Balance: " << acc.balance << " EGP\n";
    out << "------------------------------\n";
    return out;
}
    double getBalance() const { return balance; }
    string getName() const { return accountHolder; }

    static void showCounter() {
        cout << "Total Bank Accounts Created: " << counter << endl;
    }

    ~BankAccount() {
        if (accountNumber != -1)
            cout << "Account [" << accountNumber << "] of " << accountHolder << " closed.\n";
    }
};

int BankAccount::counter = 0;
int BankAccount::nextID = 1000;
```



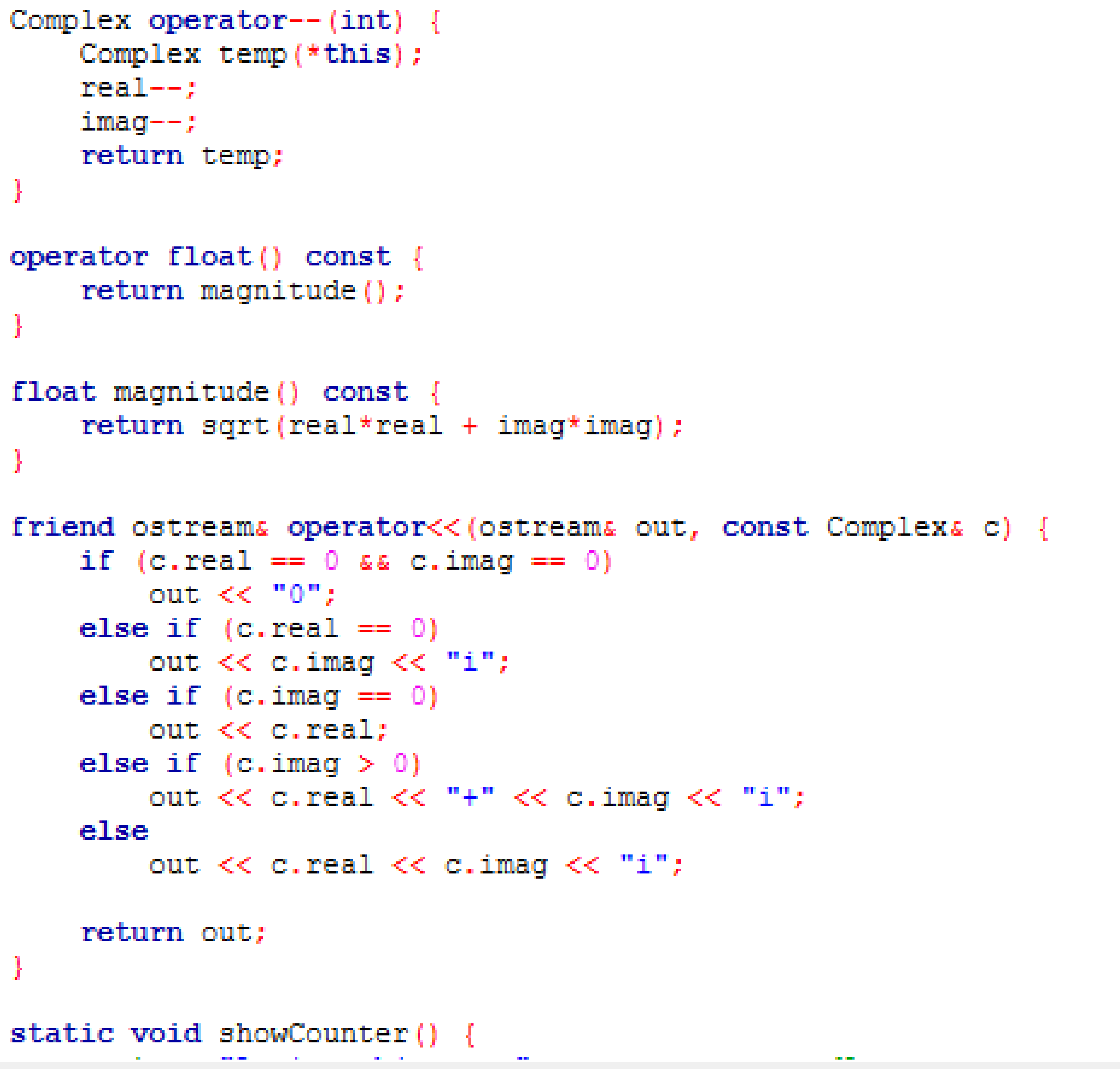
```cpp
int main() {
    cout << "--- Bank System with Move Constructor & No Copy Allowed ---\n\n";

    BankAccount acc1("Ahmed", 1200);
    BankAccount acc2("Sara", 3000);

    // Move Constructor
    cout << "\n--- Moving acc1 into acc3 ---\n";
    BankAccount acc3 = move(acc1);

 cout << "\n--- Showing Accounts ---\n";
    cout << acc2;
    cout << acc3;

    cout << "\n--- Total Accounts Created ---\n";
    BankAccount::showCounter();

    return 0;
}
```



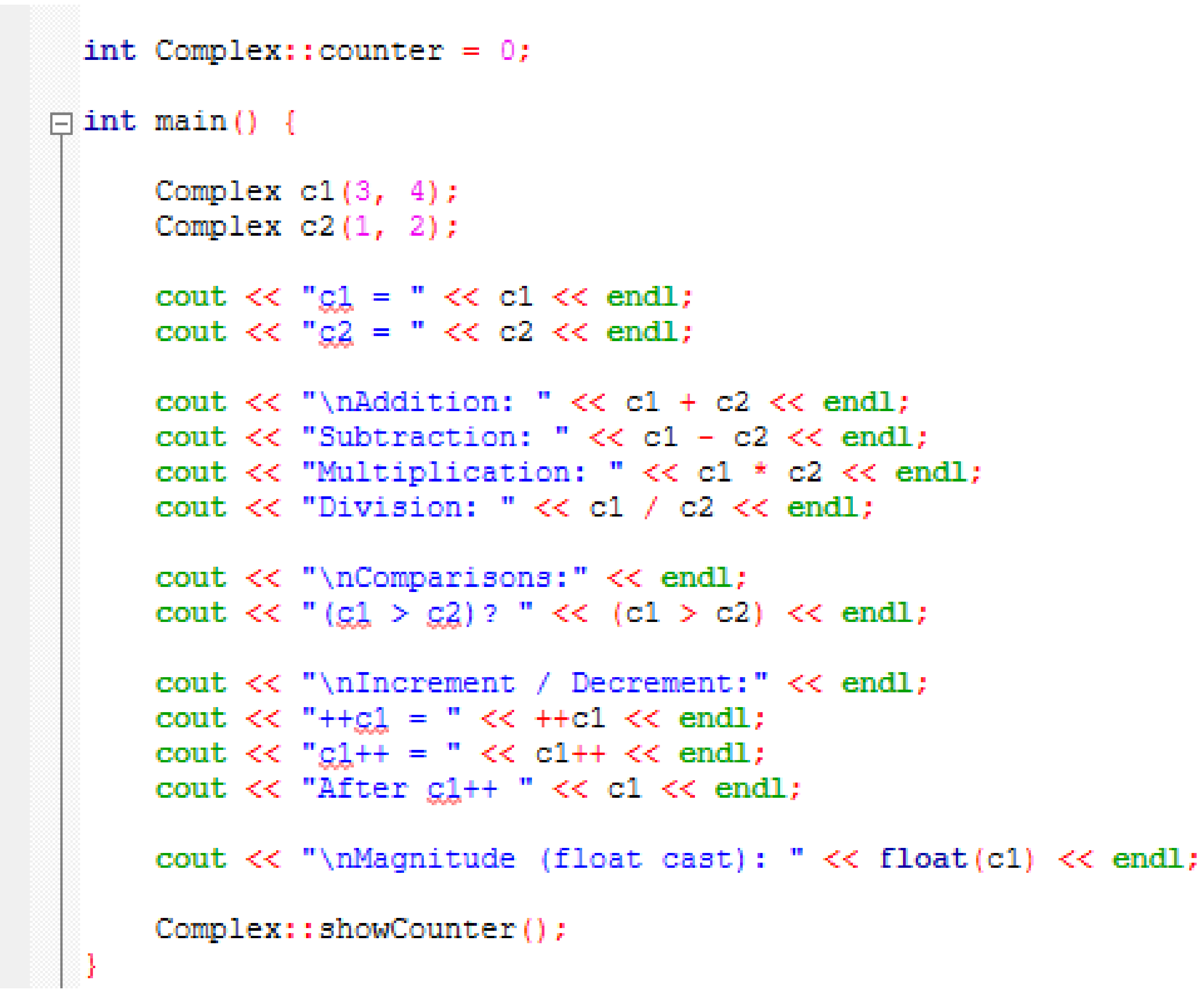
```cpp
class Complex {
private:
    float real;
    float imag;

    static int counter;

public:
    Complex() = default;

    Complex(float r, float i) : real(r), imag(i) {
        counter++;
    }

    Complex(float r) : Complex(r, 0) {
        cout << "1-parameter constructor called.\n";
    }

    ~Complex() {
        counter--;
    }

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

    Complex operator-(const Complex& c) const {
        return Complex(real - c.real, imag - c.imag);
    }

    Complex operator*(const Complex& c) const {
```



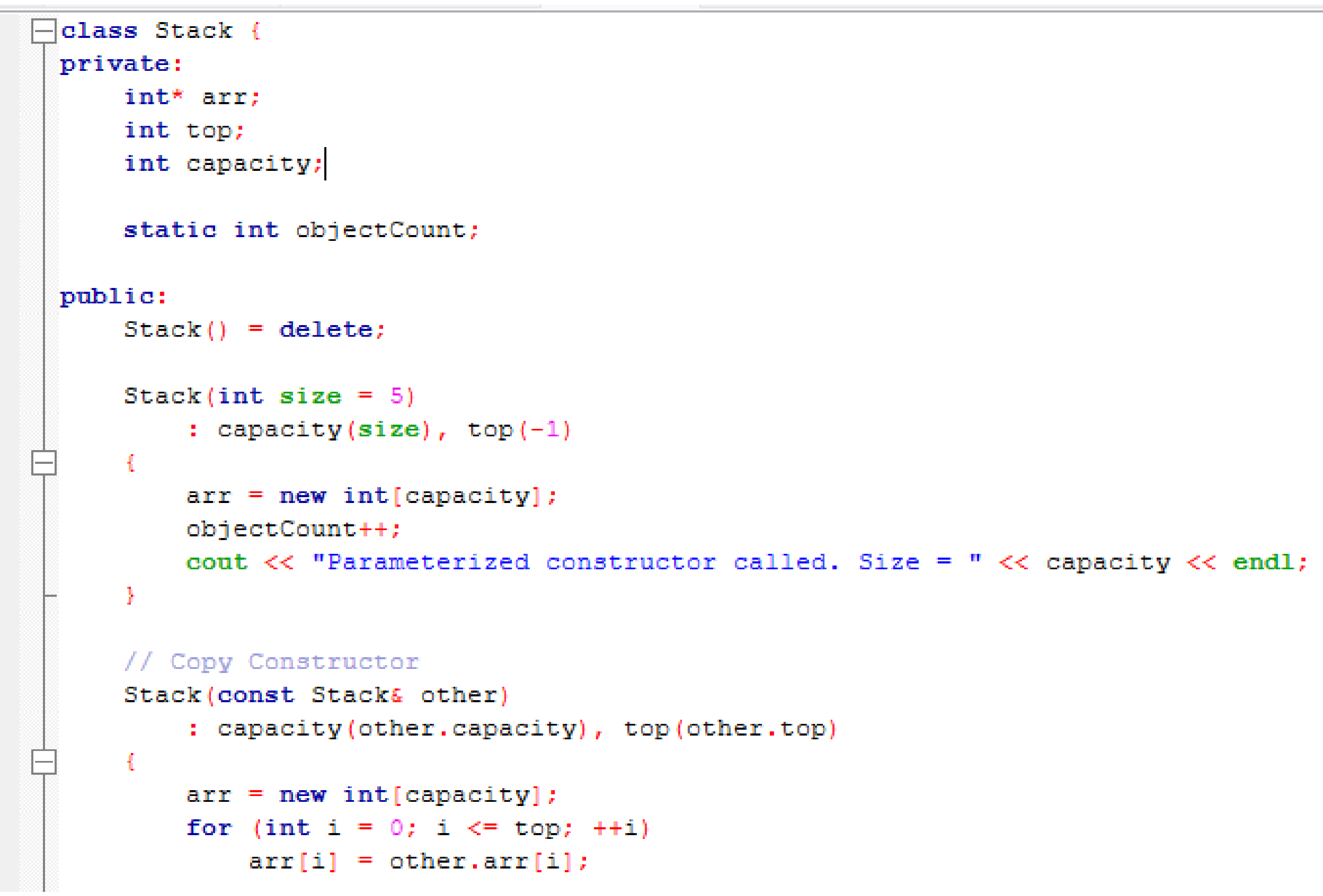
```
Complex operator/(const Complex& c) const {
    float denom = c.real * c.real + c.imag * c.imag;
    return Complex(
        (real * c.real + imag * c.imag) / denom,
        (imag * c.real - real * c.imag) / denom
    );
}

bool operator<(const Complex& c) const {
    return magnitude() < c.magnitude();
}

bool operator>(const Complex& c) const {
    return magnitude() > c.magnitude();
}

Complex& operator++() {
    real++;
    imag++;
    return *this;
}

Complex operator++(int) {
    Complex temp(*this);
    real++;
    imag++;
    return temp;
```



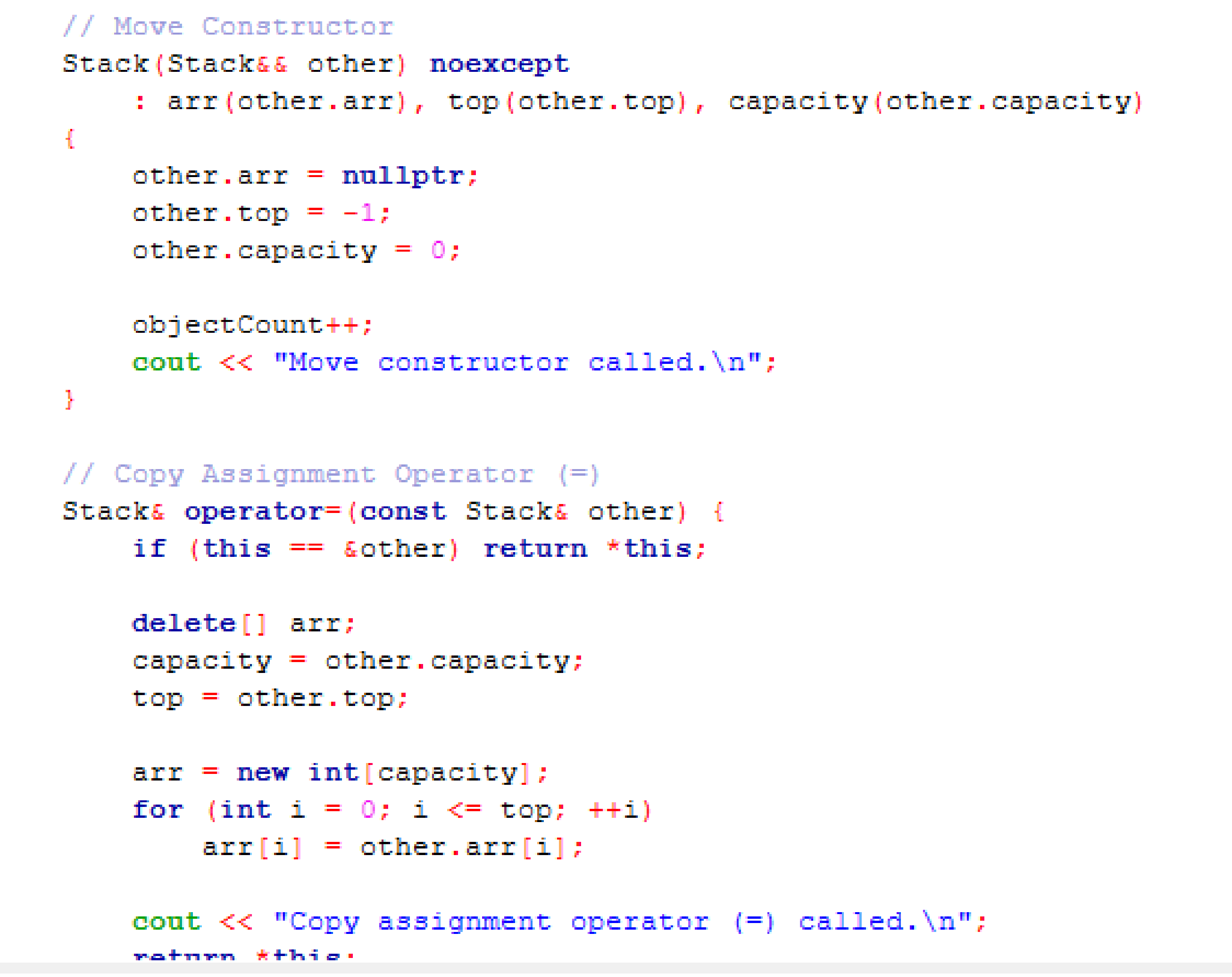
```
Complex operator--(int) {
    Complex temp(*this);
    real--;
    imag--;
    return temp;
}

operator float() const {
    return magnitude();
}

float magnitude() const {
    return sqrt(real*real + imag*imag);
}

friend ostream& operator<<(ostream& out, const Complex& c) {
    if (c.real == 0 && c.imag == 0)
        out << "0";
    else if (c.real == 0)
        out << c.imag << "i";
    else if (c.imag == 0)
        out << c.real;
    else if (c.imag > 0)
        out << c.real << "+" << c.imag << "i";
    else
        out << c.real << c.imag << "i";

    return out;
}

static void showCounter() {
```



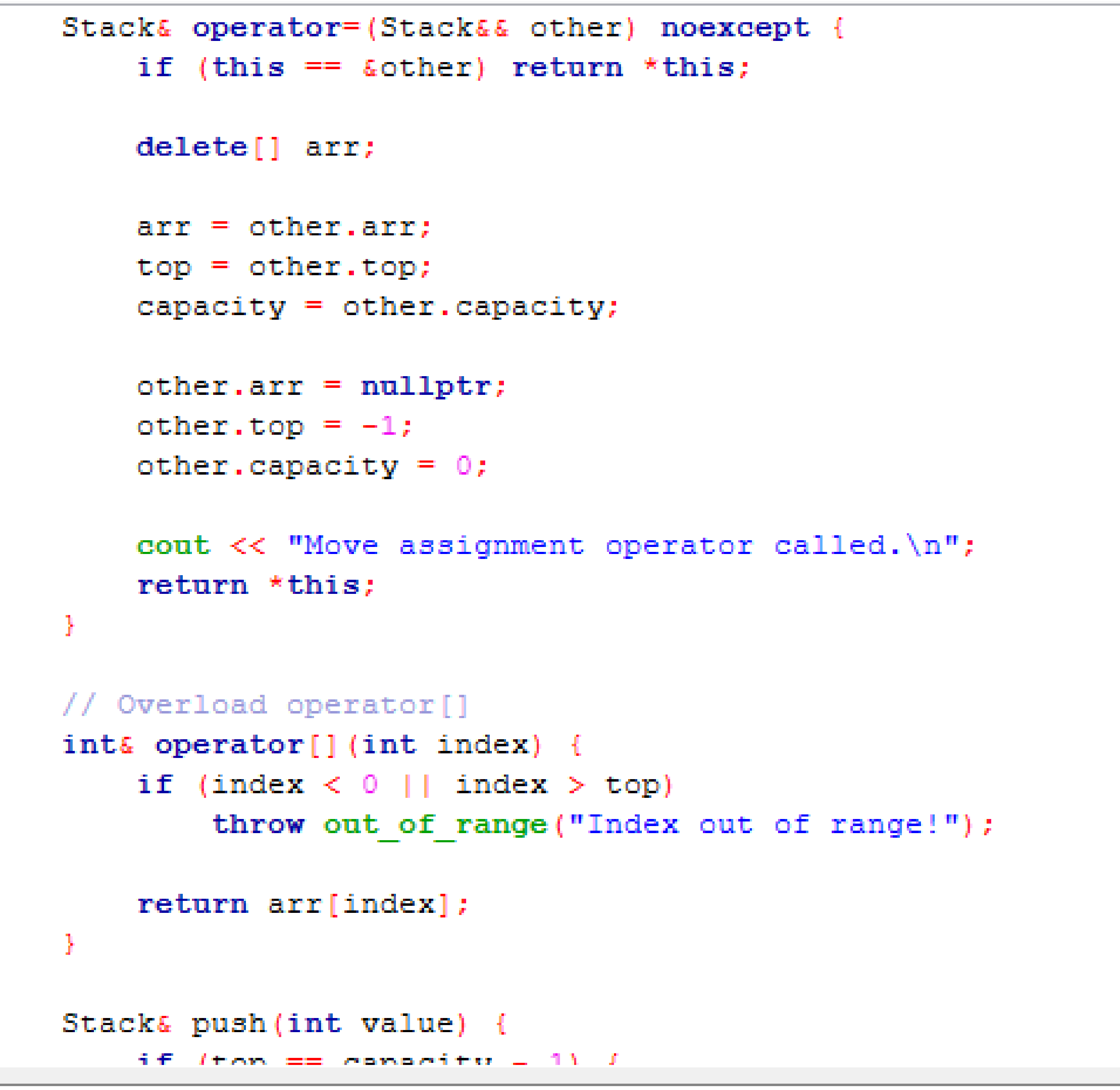
```cpp
int Complex::counter = 0;

int main() {

    Complex c1(3, 4);
    Complex c2(1, 2);

    cout << "c1 = " << c1 << endl;
    cout << "c2 = " << c2 << endl;

    cout << "\nAddition: " << c1 + c2 << endl;
    cout << "Subtraction: " << c1 - c2 << endl;
    cout << "Multiplication: " << c1 * c2 << endl;
    cout << "Division: " << c1 / c2 << endl;

    cout << "\nComparisons:" << endl;
    cout << "(c1 > c2)? " << (c1 > c2) << endl;

    cout << "\nIncrement / Decrement:" << endl;
    cout << "++c1 = " << ++c1 << endl;
    cout << "c1++ = " << c1++ << endl;
    cout << "After c1++ " << c1 << endl;

    cout << "\nMagnitude (float cast): " << float(c1) << endl;

    Complex::showCounter();
}
```



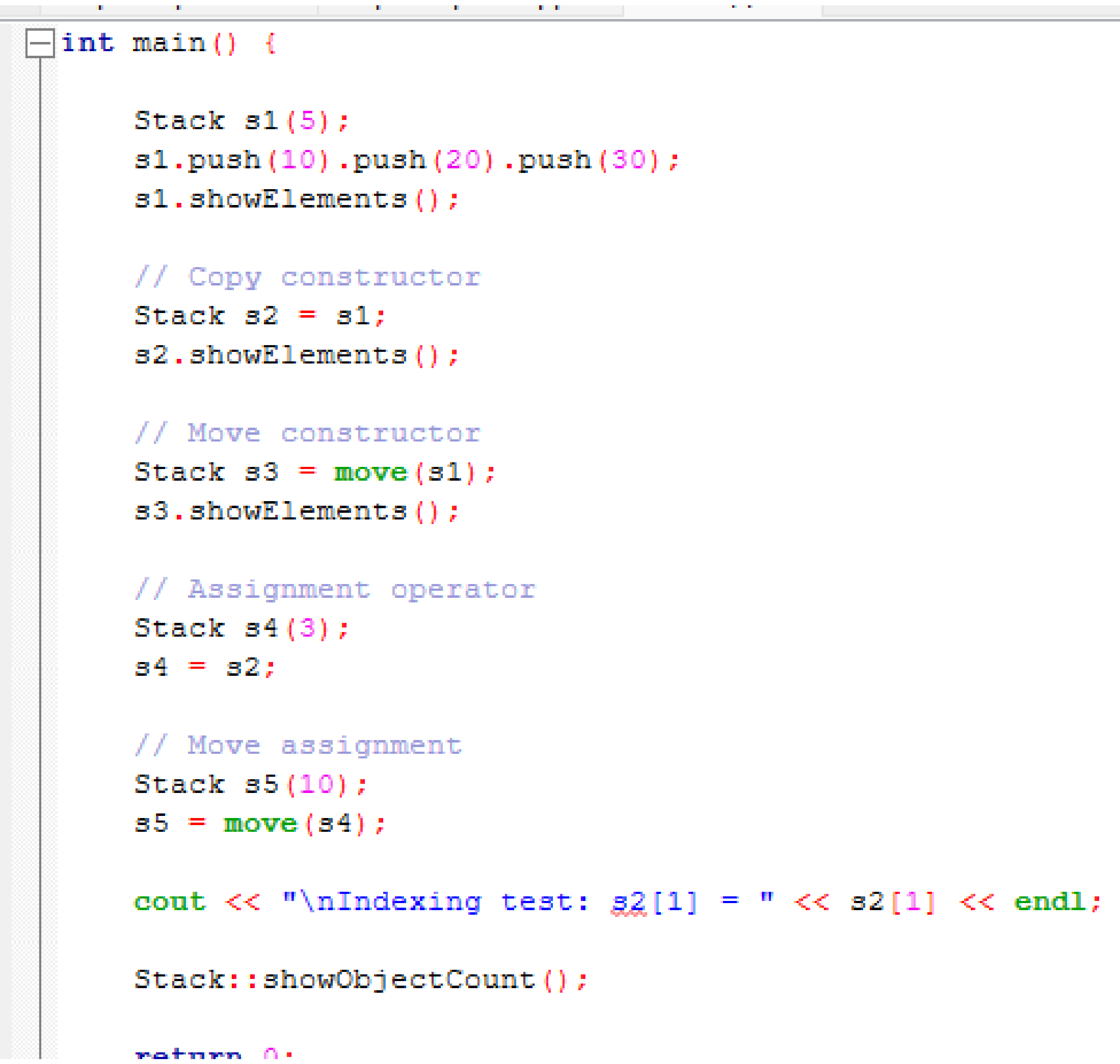
```
class Stack {
private:
    int* arr;
    int top;
    int capacity;

    static int objectCount;

public:
    Stack() = delete;

    Stack(int size = 5)
        : capacity(size), top(-1)
    {
        arr = new int[capacity];
        objectCount++;
        cout << "Parameterized constructor called. Size = " << capacity << endl;
    }

    // Copy Constructor
    Stack(const Stack& other)
        : capacity(other.capacity), top(other.top)
    {
        arr = new int[capacity];
        for (int i = 0; i <= top; ++i)
            arr[i] = other.arr[i];
```



```
// Move Constructor
Stack(Stack&& other) noexcept
    : arr(other.arr), top(other.top), capacity(other.capacity)
{
    other.arr = nullptr;
    other.top = -1;
    other.capacity = 0;

    objectCount++;
    cout << "Move constructor called.\n";
}

// Copy Assignment Operator (=)
Stack& operator=(const Stack& other) {
    if (this == &other) return *this;

    delete[] arr;
    capacity = other.capacity;
    top = other.top;

    arr = new int[capacity];
    for (int i = 0; i <= top; ++i)
        arr[i] = other.arr[i];

    cout << "Copy assignment operator (=) called.\n";
    return *this;
```



```cpp
Stack& operator=(Stack&& other) noexcept {
    if (this == &other) return *this;

    delete[] arr;

    arr = other.arr;
    top = other.top;
    capacity = other.capacity;

    other.arr = nullptr;
    other.top = -1;
    other.capacity = 0;

    cout << "Move assignment operator called.\n";
    return *this;
}

// Overload operator[]
int& operator[](int index) {
    if (index < 0 || index > top)
        throw out_of_range("Index out of range!");

    return arr[index];
}

Stack& push(int value) {
    if (top == capacity - 1) {
```



```
int main() {

    Stack s1(5);
    s1.push(10).push(20).push(30);
    s1.showElements();

    // Copy constructor
    Stack s2 = s1;
    s2.showElements();

    // Move constructor
    Stack s3 = move(s1);
    s3.showElements();

    // Assignment operator
    Stack s4(3);
    s4 = s2;

    // Move assignment
    Stack s5(10);
    s5 = move(s4);

    cout << "\nIndexing test: s2[1] = " << s2[1] << endl;

    Stack::showObjectCount();

    return 0;
```