SAVEETHA SCHOOL OF ENGINEERING

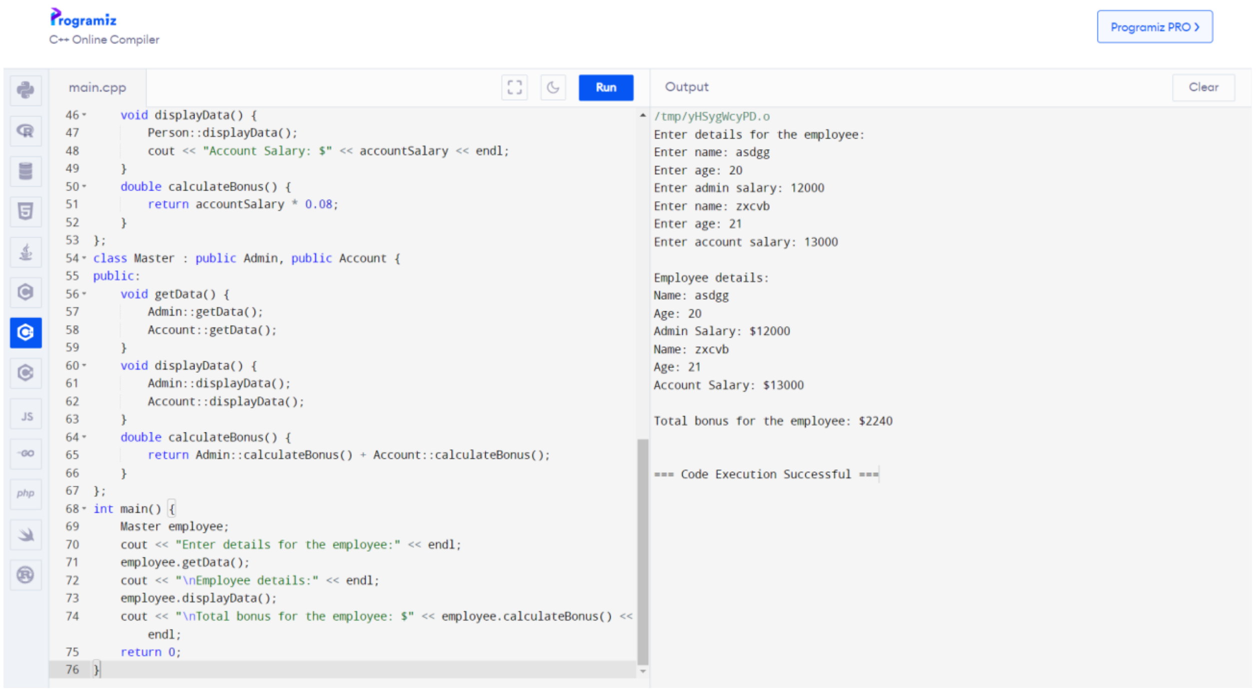
SAVEETHA INSTITUTE OF MEDICAL AND TECHNICAL SCIENCES

SUB CODE & NAME : DSA01/ Object Oriented Programming with C++

LAB DAY 5/20-03-2024

EASY

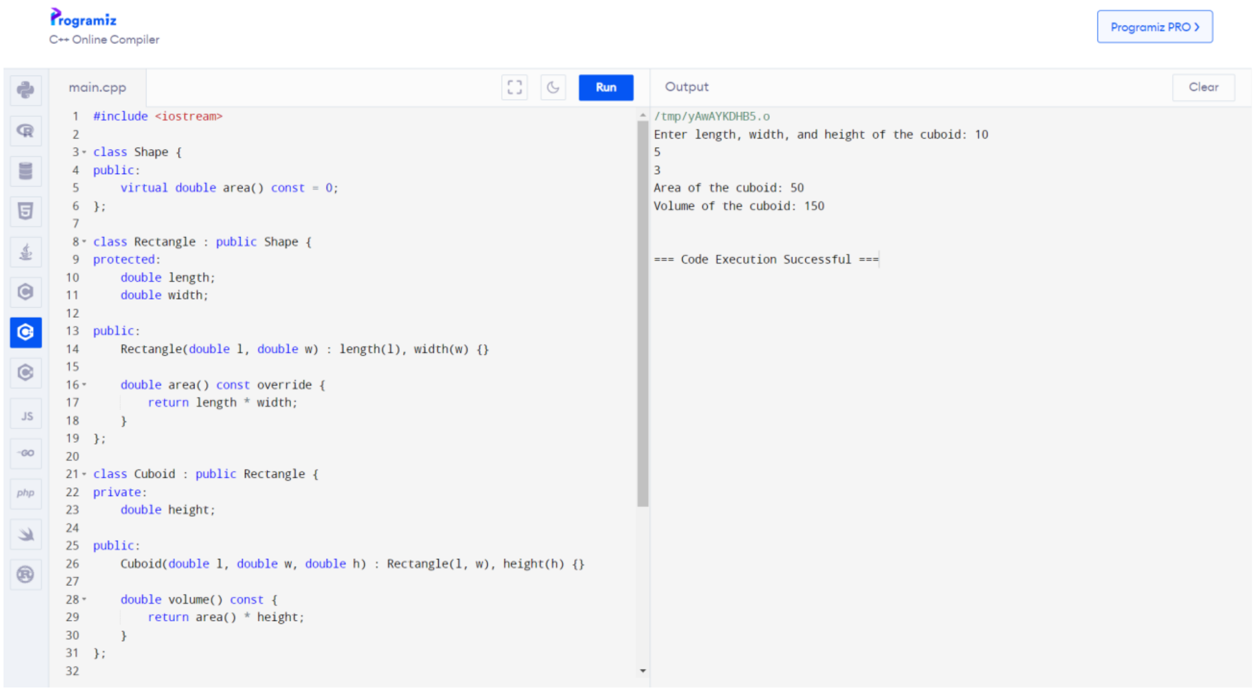
1. Write a program to calculate the bonus of the employees. The class master derives the information from both admin and account classes which derives information from the class person. Create base and all derived classes having same member functions and parameters called getdata, display data and bonus.



```cpp
    void displayData() {
        Person::displayData();
        cout << "Account Salary: $" << accountSalary << endl;
    }
    double calculateBonus() {
        return accountSalary * 0.08;
    }
};
class Master : public Admin, public Account {
public:
    void getData() {
        Admin::getData();
        Account::getData();
    }
    void displayData() {
        Admin::displayData();
        Account::displayData();
    }
    double calculateBonus() {
        return Admin::calculateBonus() + Account::calculateBonus();
    }
};
int main() {
    Master employee;
    cout << "Enter details for the employee:" << endl;
    employee.getData();
    cout << "\nEmployee details:" << endl;
    employee.displayData();
    cout << "\nTotal bonus for the employee: $" << employee.calculateBonus() <<
        endl;
    return 0;
}
```

```
/tmp/yHSygWcyPD.o
Enter details for the employee:
Enter name: asdgg
Enter age: 20
Enter admin salary: 12000
Enter name: zxcvb
Enter age: 21
Enter account salary: 13000

Employee details:
Name: asdgg
Age: 20
Admin Salary: $12000
Name: zxcvb
Age: 21
Account Salary: $13000

Total bonus for the employee: $2240


=== Code Execution Successful ===
```

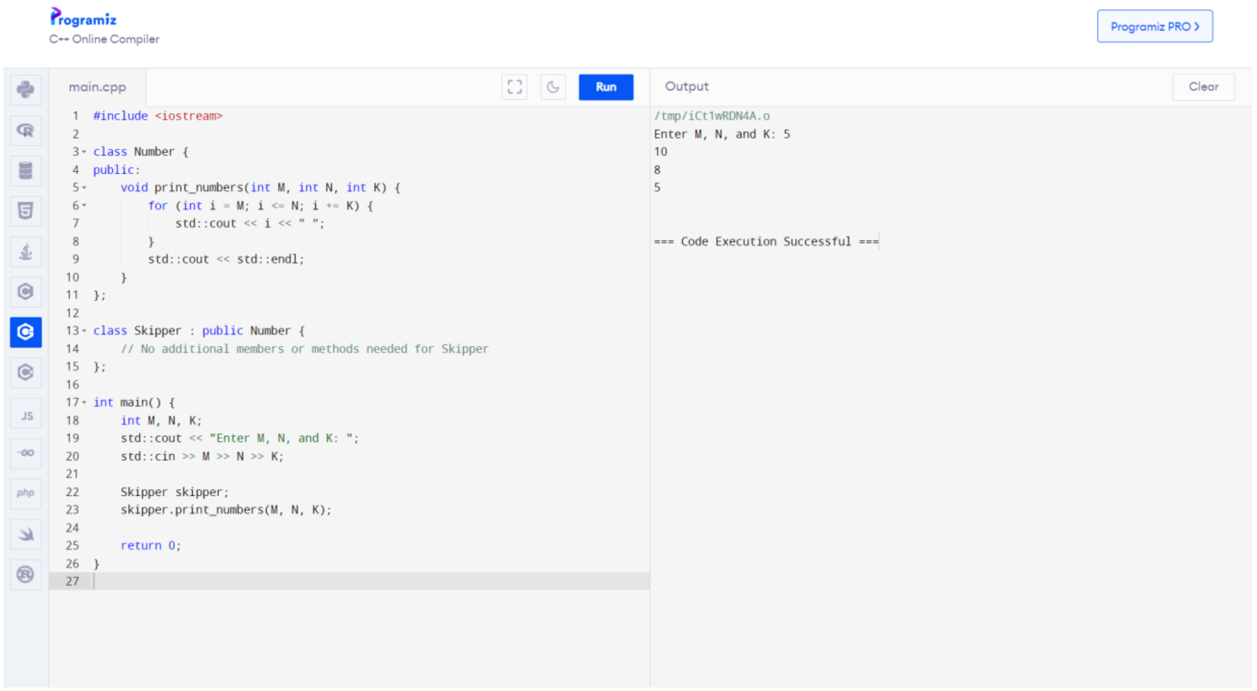
2. Write a C++ program to demonstrate multiple inheritance by creating a class cuboid which extends class rectangle, class shape. It calculates area and volume. Use appropriate member functions and member variables.



```cpp
#include <iostream>

class Shape {
public:
    virtual double area() const = 0;
};

class Rectangle : public Shape {
protected:
    double length;
    double width;

public:
    Rectangle(double l, double w) : length(l), width(w) {}

    double area() const override {
        return length * width;
    }
};

class Cuboid : public Rectangle {
private:
    double height;

public:
    Cuboid(double l, double w, double h) : Rectangle(l, w), height(h) {}

    double volume() const {
        return area() * height;
    }
};

```

```
/tmp/yAwAYKDHB5.o
Enter length, width, and height of the cuboid: 10
5
3
Area of the cuboid: 50
Volume of the cuboid: 150


=== Code Execution Successful ===
```

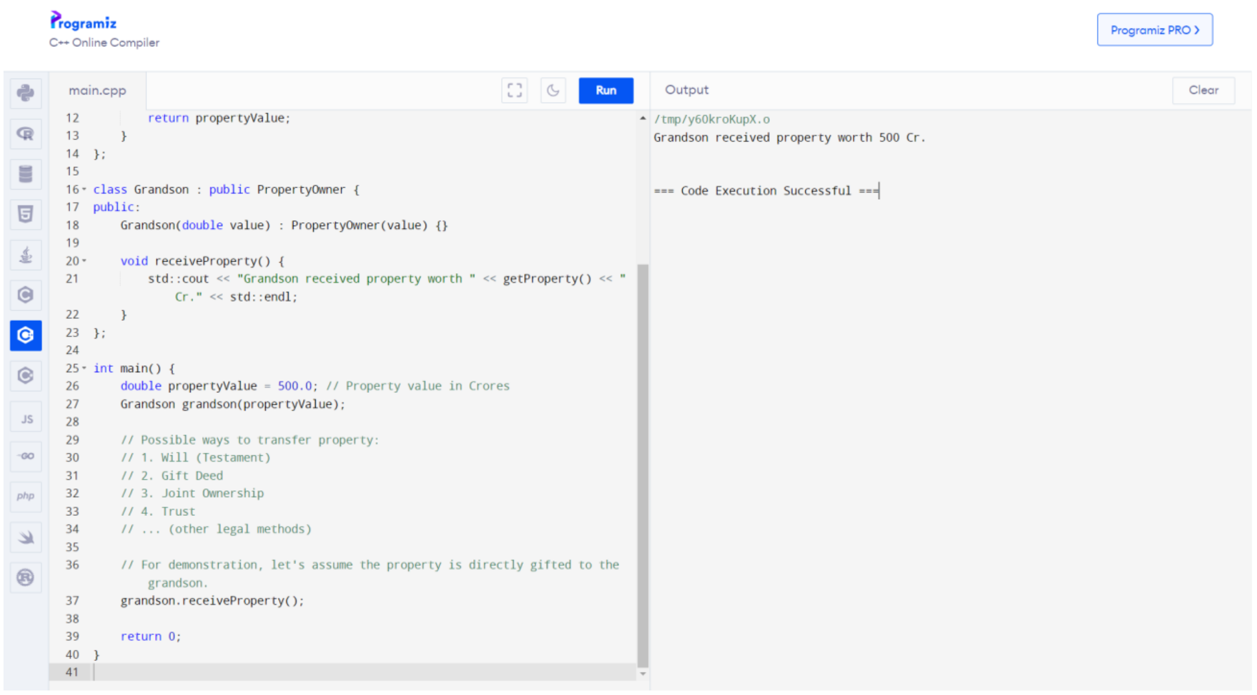
3. Create a class Number and derive the class Skipper from Number. Write a program to print the numbers from M to N by skipping K numbers in between? Use appropriate functions and variables.



```cpp
#include <iostream>

class Number {
public:
    void print_numbers(int M, int N, int K) {
        for (int i = M; i <= N; i += K) {
            std::cout << i << " ";
        }
        std::cout << std::endl;
    }
};

class Skipper : public Number {
    // No additional members or methods needed for Skipper
};

int main() {
    int M, N, K;
    std::cout << "Enter M, N, and K: ";
    std::cin >> M >> N >> K;

    Skipper skipper;
    skipper.print_numbers(M, N, K);

    return 0;
}

```

```
/tmp/iCt1wRDN4A.o
Enter M, N, and K: 5
10
8
5


=== Code Execution Successful ===
```

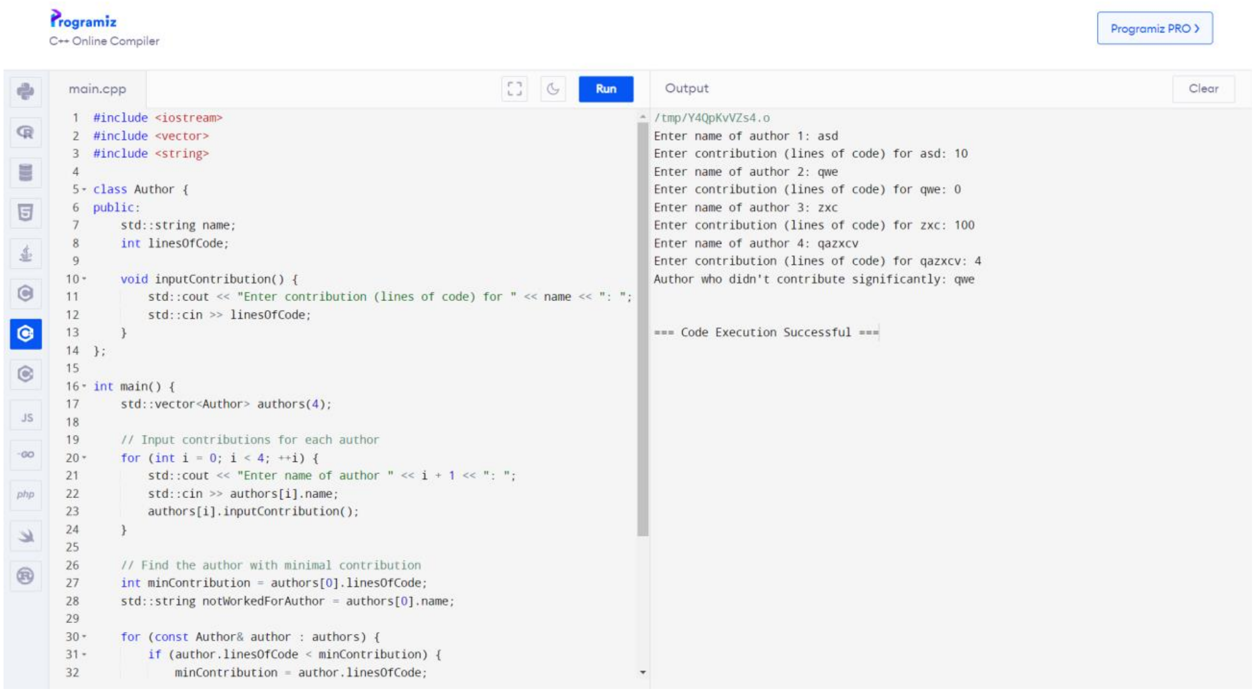
4. A Grandfather is the owner of 500Cr property, he wants to give this property to his grandson. What are the possible ways to access this property by his grandson? Use appropriate functions and variables.



```cpp
        return propertyValue;
    }
};

class Grandson : public PropertyOwner {
public:
    Grandson(double value) : PropertyOwner(value) {}

    void receiveProperty() {
        std::cout << "Grandson received property worth " << getProperty() << " Cr." << std::endl;
    }
};

int main() {
    double propertyValue = 500.0; // Property value in Crores
    Grandson grandson(propertyValue);

    // Possible ways to transfer property:
    // 1. Will (Testament)
    // 2. Gift Deed
    // 3. Joint Ownership
    // 4. Trust
    // ... (other legal methods)

    // For demonstration, let's assume the property is directly gifted to the grandson.
    grandson.receiveProperty();

    return 0;
}
```

```
/tmp/y60kroKupX.o
Grandson received property worth 500 Cr.

=== Code Execution Successful ===
```

5. A paper consists of 4 authors, but one author didn't do any work but he wants to put his name in this paper. But others are not interested at the same time they want to add another author as a 5th author. How to identify the Not worked for paper. Write a C++ code for the above scenario.



```cpp
#include <iostream>
#include <vector>
#include <string>

class Author {
public:
    std::string name;
    int linesOfCode;

    void inputContribution() {
        std::cout << "Enter contribution (lines of code) for " << name << ": ";
        std::cin >> linesOfCode;
    }
};

int main() {
    std::vector<Author> authors(4);

    // Input contributions for each author
    for (int i = 0; i < 4; ++i) {
        std::cout << "Enter name of author " << i + 1 << ": ";
        std::cin >> authors[i].name;
        authors[i].inputContribution();
    }

    // Find the author with minimal contribution
    int minContribution = authors[0].linesOfCode;
    std::string notWorkedForAuthor = authors[0].name;

    for (const Author& author : authors) {
        if (author.linesOfCode < minContribution) {
            minContribution = author.linesOfCode;
```

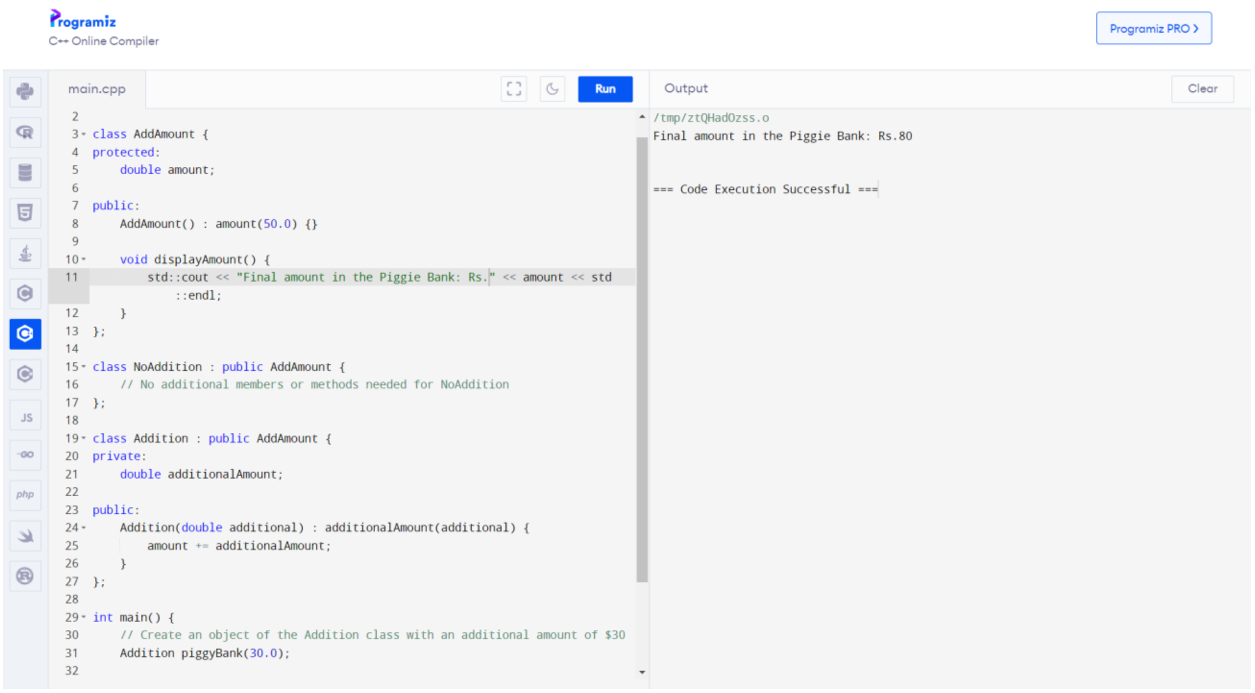
```
/tmp/Y4QpKvVZs4.o
Enter name of author 1: asd
Enter contribution (lines of code) for asd: 10
Enter name of author 2: qwe
Enter contribution (lines of code) for qwe: 0
Enter name of author 3: zxc
Enter contribution (lines of code) for zxc: 100
Enter name of author 4: qazxcv
Enter contribution (lines of code) for qazxcv: 4
Author who didn't contribute significantly: qwe

=== Code Execution Successful ===
```

MEDIUM

1. Suppose you have a Piggie Bank with an initial amount of $50 and you have to add some more amount to it. Create a class 'AddAmount' with a data member named 'amount' with an initial value of $50. Now make two other derived class as follows:

Base class - no amount will be added to the Piggie Bank

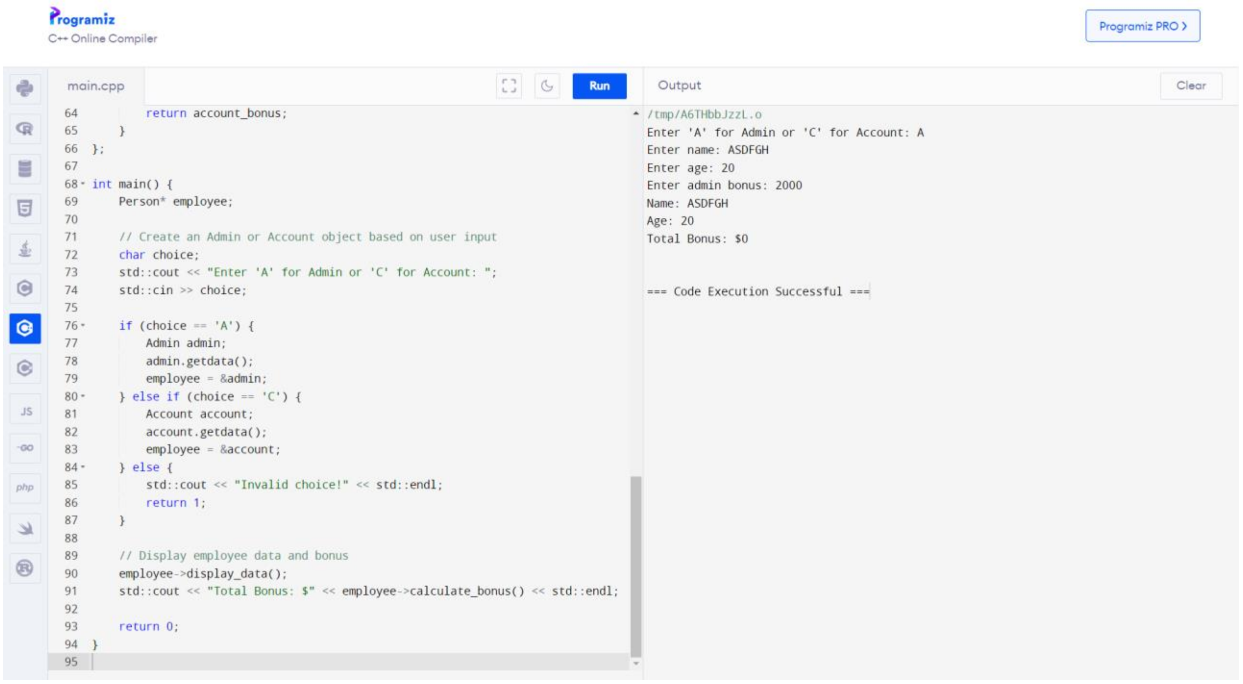
Derived class - having a parameter which is the amount that will be added to the Piggie Bank Create an object of the 'AddAmount' class and display the final amount in the Piggie Bank.



```cpp

class AddAmount {
protected:
    double amount;

public:
    AddAmount() : amount(50.0) {}

    void displayAmount() {
        std::cout << "Final amount in the Piggie Bank: Rs." << amount << std
            ::endl;
    }
};

class NoAddition : public AddAmount {
    // No additional members or methods needed for NoAddition
};

class Addition : public AddAmount {
private:
    double additionalAmount;

public:
    Addition(double additional) : additionalAmount(additional) {
        amount += additionalAmount;
    }
};

int main() {
    // Create an object of the Addition class with an additional amount of $30
    Addition piggyBank(30.0);

```

```
/tmp/ztQHadOzss.o
Final amount in the Piggie Bank: Rs.80


=== Code Execution Successful ===
```

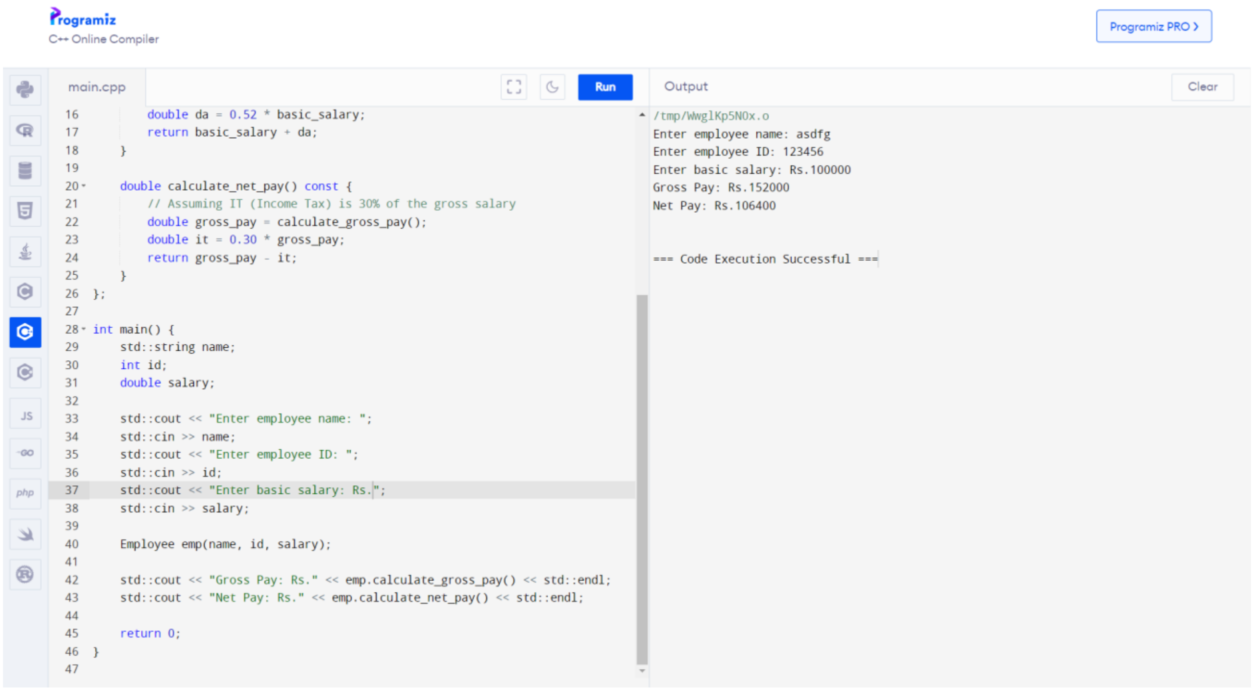
2. Write a program to calculate the bonus of the employees. The class master derives the information from both admin and account classes which derives information from the class person. Create base and all derived classes having same member functions and different parameters called getdata, display data and bonus. Create a base class pointer that is capable of accessing data of any class and calculates the bonus of the specified employee.



```cpp
        return account_bonus;
    }
};

int main() {
    Person* employee;

    // Create an Admin or Account object based on user input
    char choice;
    std::cout << "Enter 'A' for Admin or 'C' for Account: ";
    std::cin >> choice;

    if (choice == 'A') {
        Admin admin;
        admin.getdata();
        employee = &admin;
    } else if (choice == 'C') {
        Account account;
        account.getdata();
        employee = &account;
    } else {
        std::cout << "Invalid choice!" << std::endl;
        return 1;
    }

    // Display employee data and bonus
    employee->display_data();
    std::cout << "Total Bonus: $" << employee->calculate_bonus() << std::endl;

    return 0;
}
```

```
/tmp/A6THbbJzzL.o
Enter 'A' for Admin or 'C' for Account: A
Enter name: ASDFGH
Enter age: 20
Enter admin bonus: 2000
Name: ASDFGH
Age: 20
Total Bonus: $0


=== Code Execution Successful ===
```

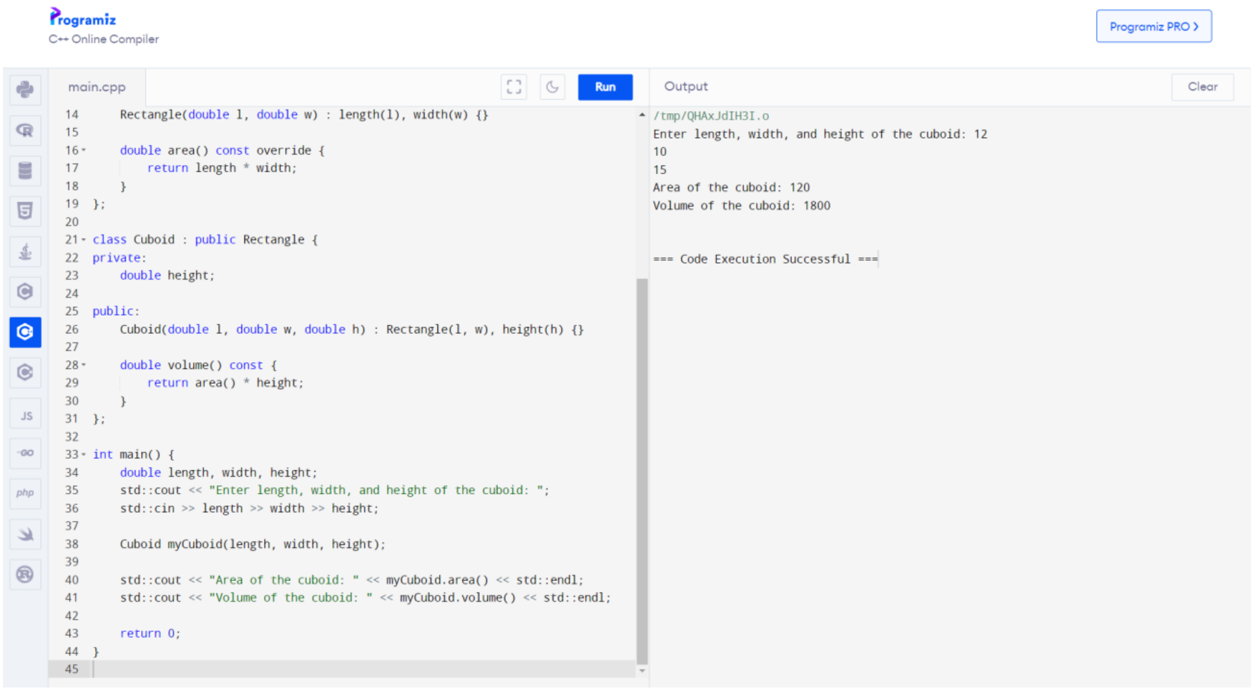
3. Write a C++ program to calculate the gross and net pay of employees from basic salary. Create an employee which consists of employee name,emp_id, and basic salary as its data members. Use parameterized constructions in the derived class to initialize data members of the base class and calculate gross and net pay of the employee in the derived class.



```cpp
        double da = 0.52 * basic_salary;
        return basic_salary + da;
    }

    double calculate_net_pay() const {
        // Assuming IT (Income Tax) is 30% of the gross salary
        double gross_pay = calculate_gross_pay();
        double it = 0.30 * gross_pay;
        return gross_pay - it;
    }
};

int main() {
    std::string name;
    int id;
    double salary;

    std::cout << "Enter employee name: ";
    std::cin >> name;
    std::cout << "Enter employee ID: ";
    std::cin >> id;
    std::cout << "Enter basic salary: Rs.";
    std::cin >> salary;

    Employee emp(name, id, salary);

    std::cout << "Gross Pay: Rs." << emp.calculate_gross_pay() << std::endl;
    std::cout << "Net Pay: Rs." << emp.calculate_net_pay() << std::endl;

    return 0;
}
```

```
/tmp/WwglKp5N0x.o
Enter employee name: asdfg
Enter employee ID: 123456
Enter basic salary: Rs.100000
Gross Pay: Rs.152000
Net Pay: Rs.106400


=== Code Execution Successful ===
```

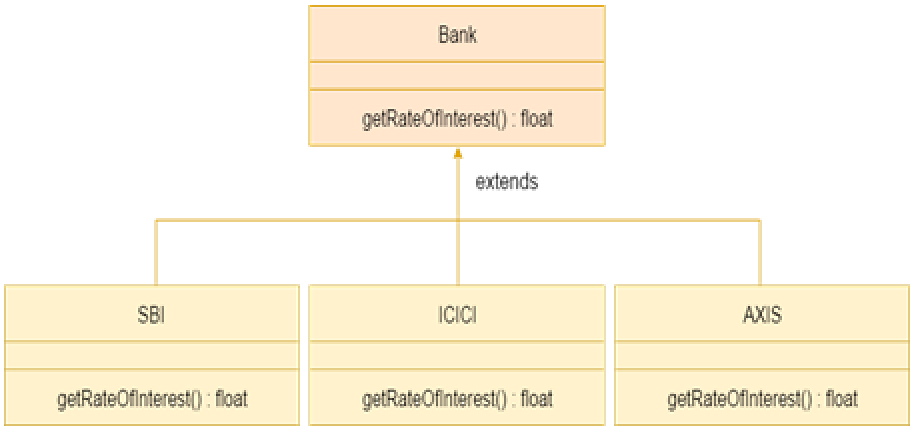
4. Write a C++ program to demonstrate the multiple inheritance by creating a class cuboid which extends class rectangle, class shape. It calculates area and volume.

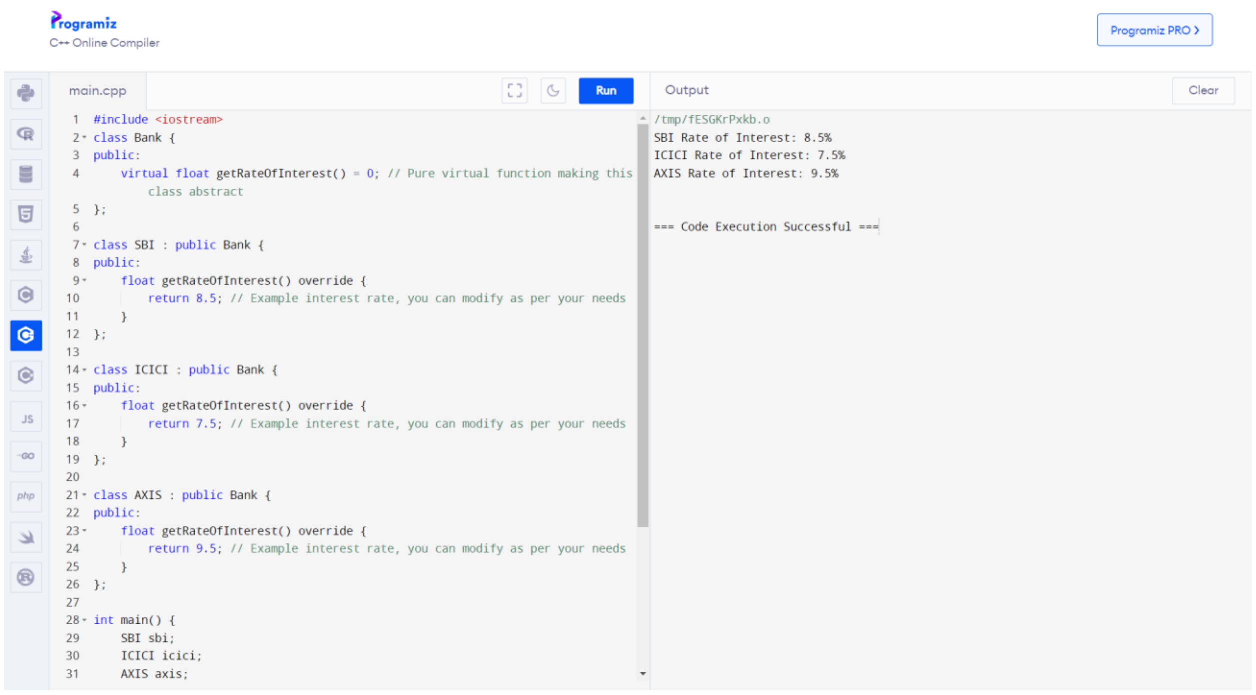


```cpp
    Rectangle(double l, double w) : length(l), width(w) {}

    double area() const override {
        return length * width;
    }
};

class Cuboid : public Rectangle {
private:
    double height;

public:
    Cuboid(double l, double w, double h) : Rectangle(l, w), height(h) {}

    double volume() const {
        return area() * height;
    }
};

int main() {
    double length, width, height;
    std::cout << "Enter length, width, and height of the cuboid: ";
    std::cin >> length >> width >> height;

    Cuboid myCuboid(length, width, height);

    std::cout << "Area of the cuboid: " << myCuboid.area() << std::endl;
    std::cout << "Volume of the cuboid: " << myCuboid.volume() << std::endl;

    return 0;
}
```

```
/tmp/QHAxJdIH3I.o
Enter length, width, and height of the cuboid: 12
10
15
Area of the cuboid: 120
Volume of the cuboid: 1800


=== Code Execution Successful ===
```

5. Write Down the Code for Following Diagram Using Inheritance


Bank
getRateOfInterest() : float
extends
SBI
getRateOfInterest() : float
ICICI
getRateOfInterest() : float
AXIS
getRateOfInterest() : float



```cpp
#include <iostream>
class Bank {
public:
    virtual float getRateOfInterest() = 0; // Pure virtual function making this
        class abstract
};

class SBI : public Bank {
public:
    float getRateOfInterest() override {
        return 8.5; // Example interest rate, you can modify as per your needs
    }
};

class ICICI : public Bank {
public:
    float getRateOfInterest() override {
        return 7.5; // Example interest rate, you can modify as per your needs
    }
};

class AXIS : public Bank {
public:
    float getRateOfInterest() override {
        return 9.5; // Example interest rate, you can modify as per your needs
    }
};

int main() {
    SBI sbi;
    ICICI icici;
    AXIS axis;
```

```
/tmp/fESGKrPxkb.o
SBI Rate of Interest: 8.5%
ICICI Rate of Interest: 7.5%
AXIS Rate of Interest: 9.5%


=== Code Execution Successful ===
```

HARD

1. Write a c++ program to calculate tax given the following conditions:

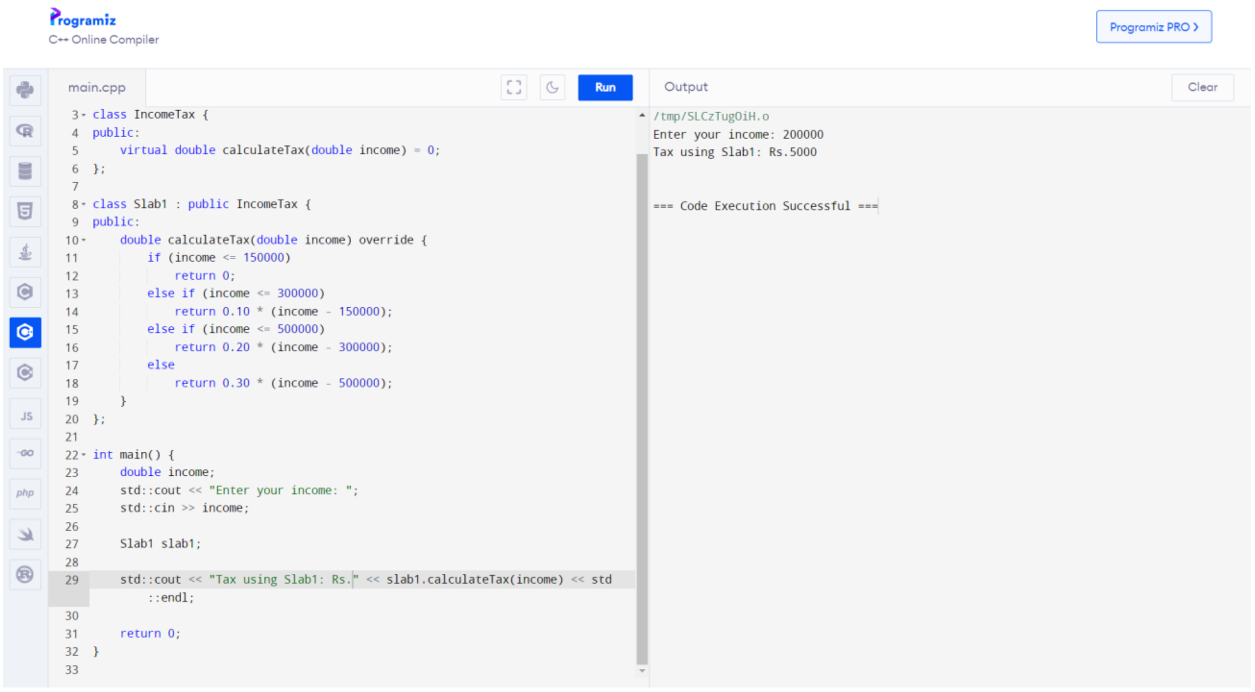
If income is less than or equal to 1,50,000 then no tax

If taxable income is 1,50,001 – 3,00,000 the charge 10% tax

If taxable income is 3,00,001 – 5,00,000 the charge 20% tax

If taxable income is above 5,00,001 then charge 30% tax

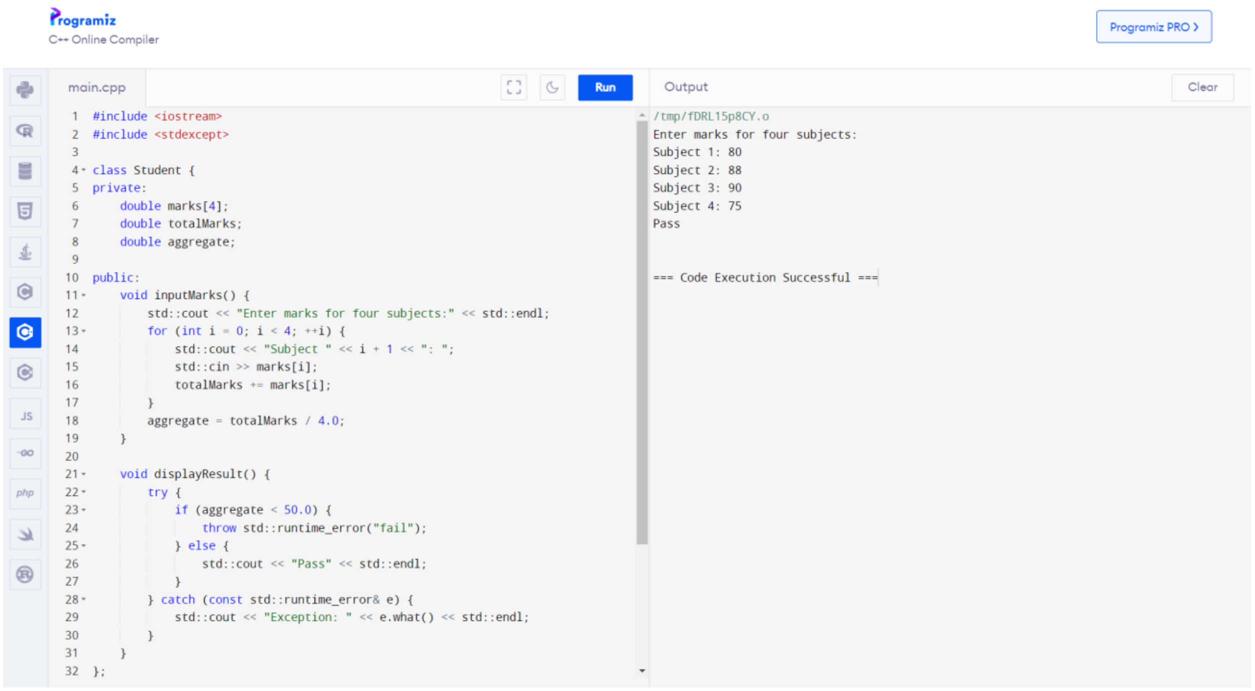
Use base class name as IncomeTax and Derived classes are Slab1, Slab 2 and Slab 3. Use TDS() all the classes.



```cpp
class IncomeTax {
public:
    virtual double calculateTax(double income) = 0;
};

class Slab1 : public IncomeTax {
public:
    double calculateTax(double income) override {
        if (income <= 150000)
            return 0;
        else if (income <= 300000)
            return 0.10 * (income - 150000);
        else if (income <= 500000)
            return 0.20 * (income - 300000);
        else
            return 0.30 * (income - 500000);
    }
};

int main() {
    double income;
    std::cout << "Enter your income: ";
    std::cin >> income;

    Slab1 slab1;

    std::cout << "Tax using Slab1: Rs." << slab1.calculateTax(income) << std
        ::endl;

    return 0;
}
```

```
/tmp/SLCzTugOiH.o
Enter your income: 200000
Tax using Slab1: Rs.5000


=== Code Execution Successful ===
```

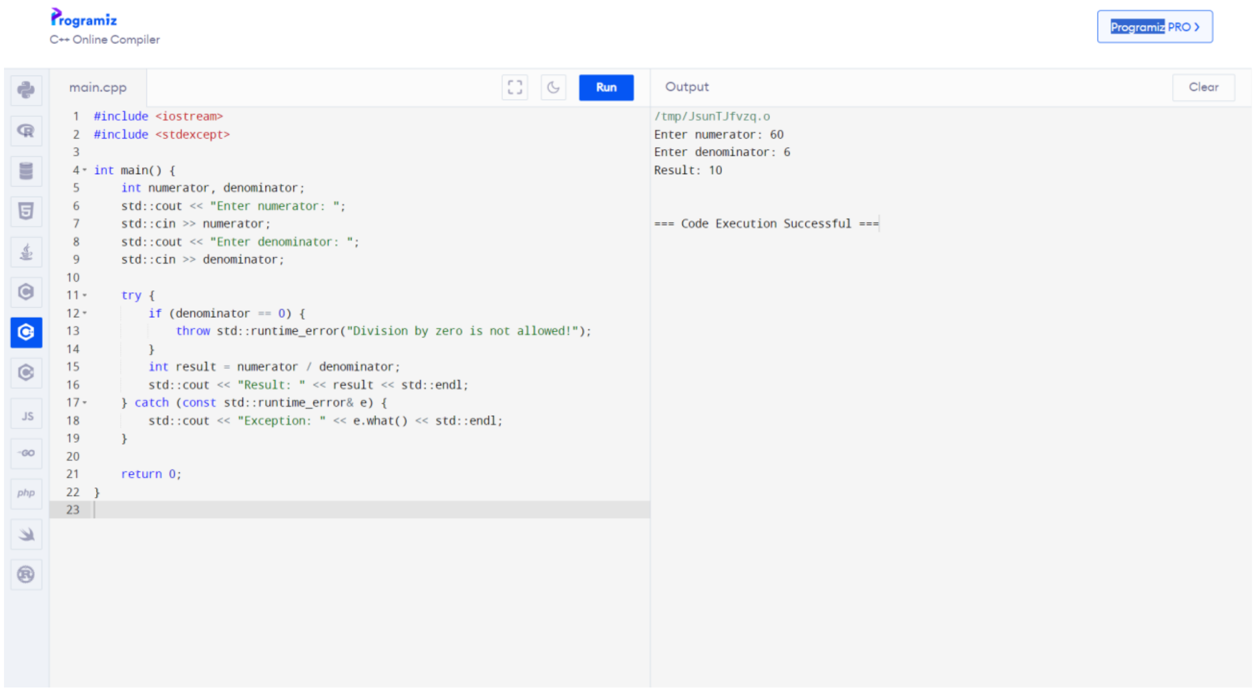
2. Write a program to enter the marks of a student in four subjects. Then calculate the total and aggregate, display the grade obtained by the student. If the student scores less than the 50 generate the exception as “fail” otherwise grade is “pass”



```cpp
#include <iostream>
#include <stdexcept>

class Student {
private:
    double marks[4];
    double totalMarks;
    double aggregate;

public:
    void inputMarks() {
        std::cout << "Enter marks for four subjects:" << std::endl;
        for (int i = 0; i < 4; ++i) {
            std::cout << "Subject " << i + 1 << ": ";
            std::cin >> marks[i];
            totalMarks += marks[i];
        }
        aggregate = totalMarks / 4.0;
    }

    void displayResult() {
        try {
            if (aggregate < 50.0) {
                throw std::runtime_error("fail");
            } else {
                std::cout << "Pass" << std::endl;
            }
        } catch (const std::runtime_error& e) {
            std::cout << "Exception: " << e.what() << std::endl;
        }
    }
};
```

```
/tmp/fDRL15p8CY.o
Enter marks for four subjects:
Subject 1: 80
Subject 2: 88
Subject 3: 90
Subject 4: 75
Pass


=== Code Execution Successful ===
```

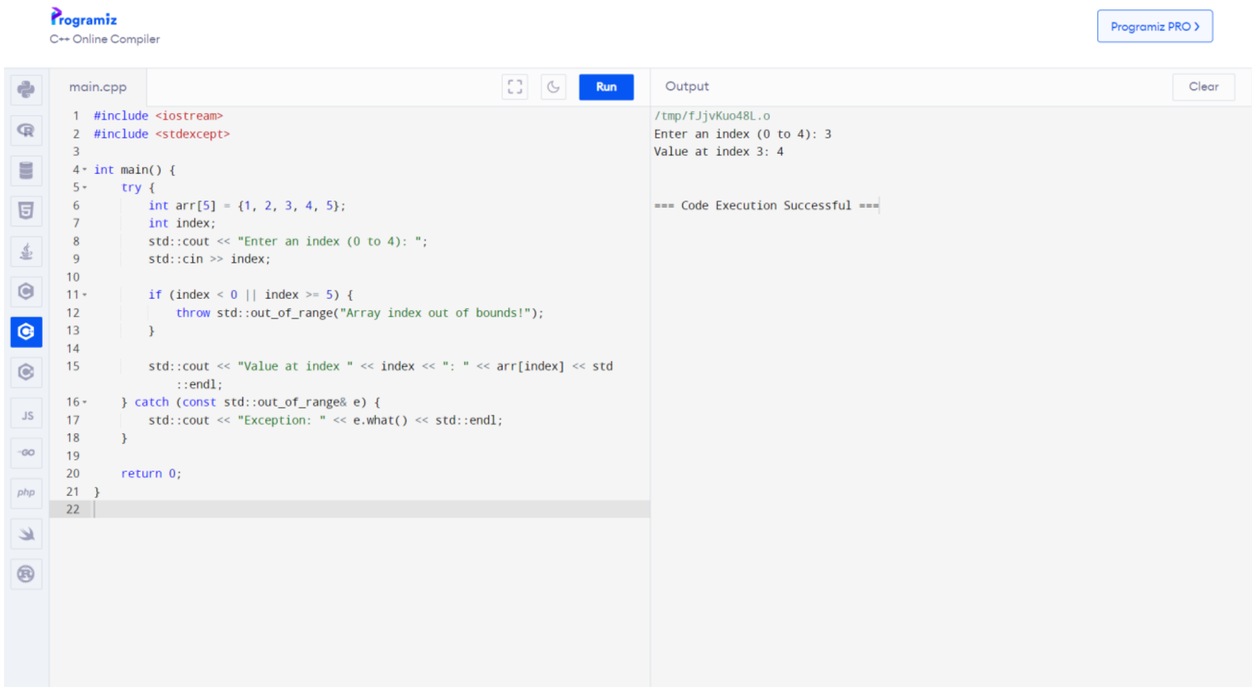
3. Write a program to illustrate division by zero exception, get the two inputs from the user, use divide (int, int).



```cpp
#include <iostream>
#include <stdexcept>

int main() {
    int numerator, denominator;
    std::cout << "Enter numerator: ";
    std::cin >> numerator;
    std::cout << "Enter denominator: ";
    std::cin >> denominator;

    try {
        if (denominator == 0) {
            throw std::runtime_error("Division by zero is not allowed!");
        }
        int result = numerator / denominator;
        std::cout << "Result: " << result << std::endl;
    } catch (const std::runtime_error& e) {
        std::cout << "Exception: " << e.what() << std::endl;
    }

    return 0;
}
```

```
/tmp/JsunTJfvzq.o
Enter numerator: 60
Enter denominator: 6
Result: 10


=== Code Execution Successful ===
```

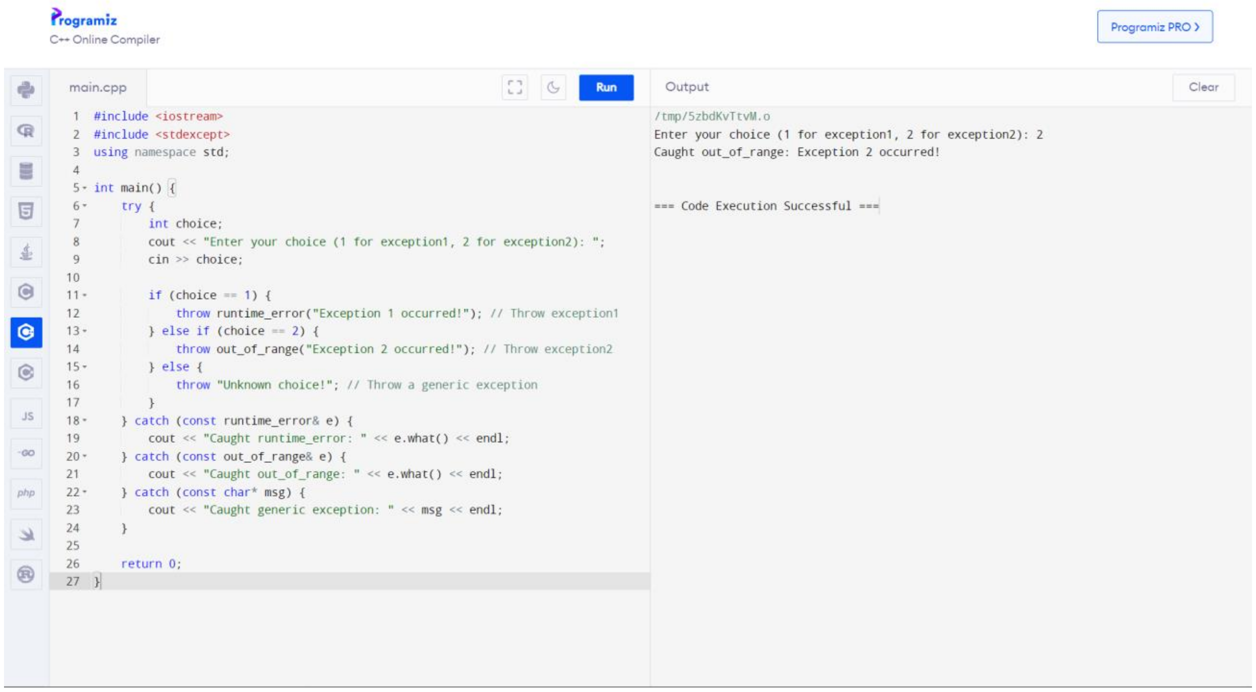
4. Write a program to illustrate array index out of bounds exceptions.



```cpp
#include <iostream>
#include <stdexcept>

int main() {
    try {
        int arr[5] = {1, 2, 3, 4, 5};
        int index;
        std::cout << "Enter an index (0 to 4): ";
        std::cin >> index;

        if (index < 0 || index >= 5) {
            throw std::out_of_range("Array index out of bounds!");
        }

        std::cout << "Value at index " << index << ": " << arr[index] << std
            ::endl;
    } catch (const std::out_of_range& e) {
        std::cout << "Exception: " << e.what() << std::endl;
    }

    return 0;
}
```

```
/tmp/fJjvKuo48L.o
Enter an index (0 to 4): 3
Value at index 3: 4


=== Code Execution Successful ===
```

5. Write a C++ program to throw multiple exceptions and define multiple catch statement.



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

int main() {
    try {
        int choice;
        cout << "Enter your choice (1 for exception1, 2 for exception2): ";
        cin >> choice;

        if (choice == 1) {
            throw runtime_error("Exception 1 occurred!"); // Throw exception1
        } else if (choice == 2) {
            throw out_of_range("Exception 2 occurred!"); // Throw exception2
        } else {
            throw "Unknown choice!"; // Throw a generic exception
        }
    } catch (const runtime_error& e) {
        cout << "Caught runtime_error: " << e.what() << endl;
    } catch (const out_of_range& e) {
        cout << "Caught out_of_range: " << e.what() << endl;
    } catch (const char* msg) {
        cout << "Caught generic exception: " << msg << endl;
    }

    return 0;
}
```

```
/tmp/5zbdKvTtvM.o
Enter your choice (1 for exception1, 2 for exception2): 2
Caught out_of_range: Exception 2 occurred!


=== Code Execution Successful ===
```