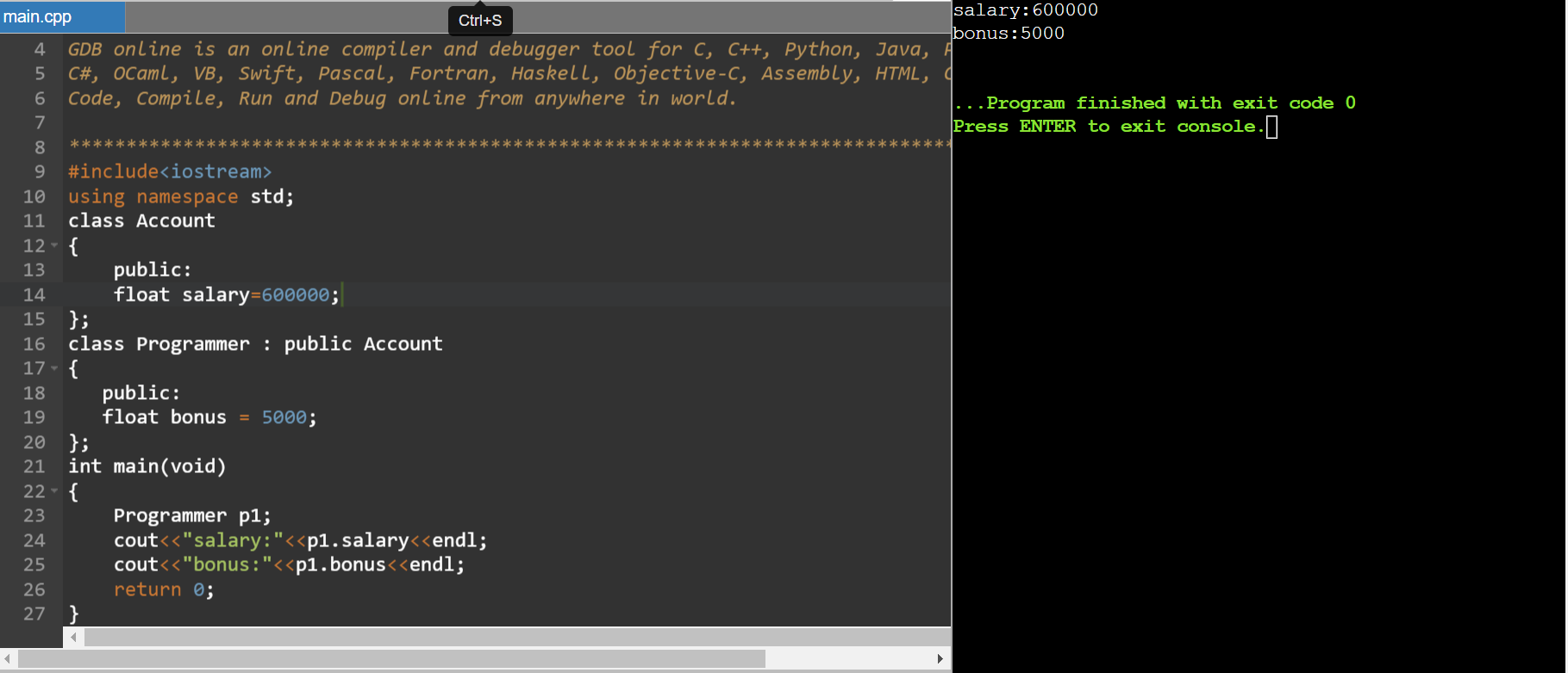
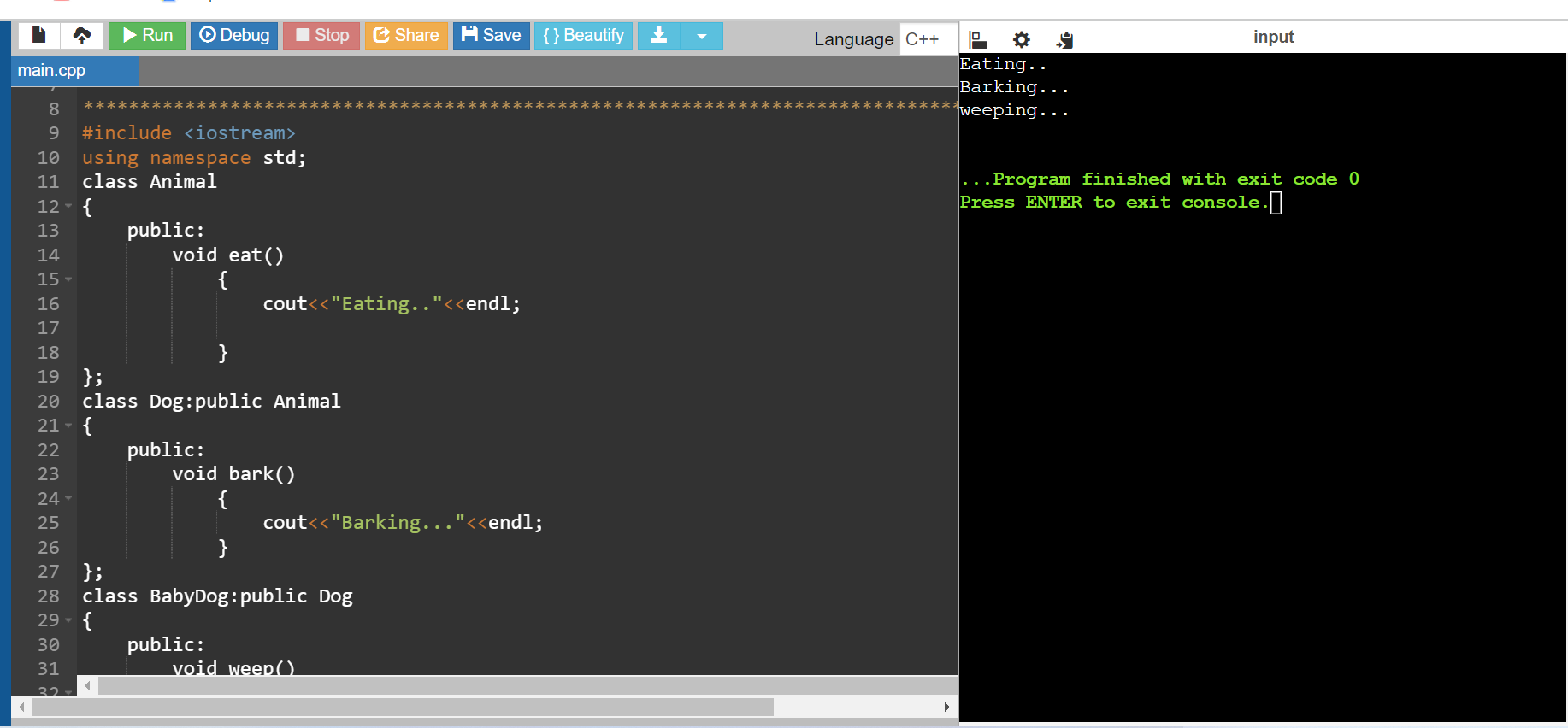
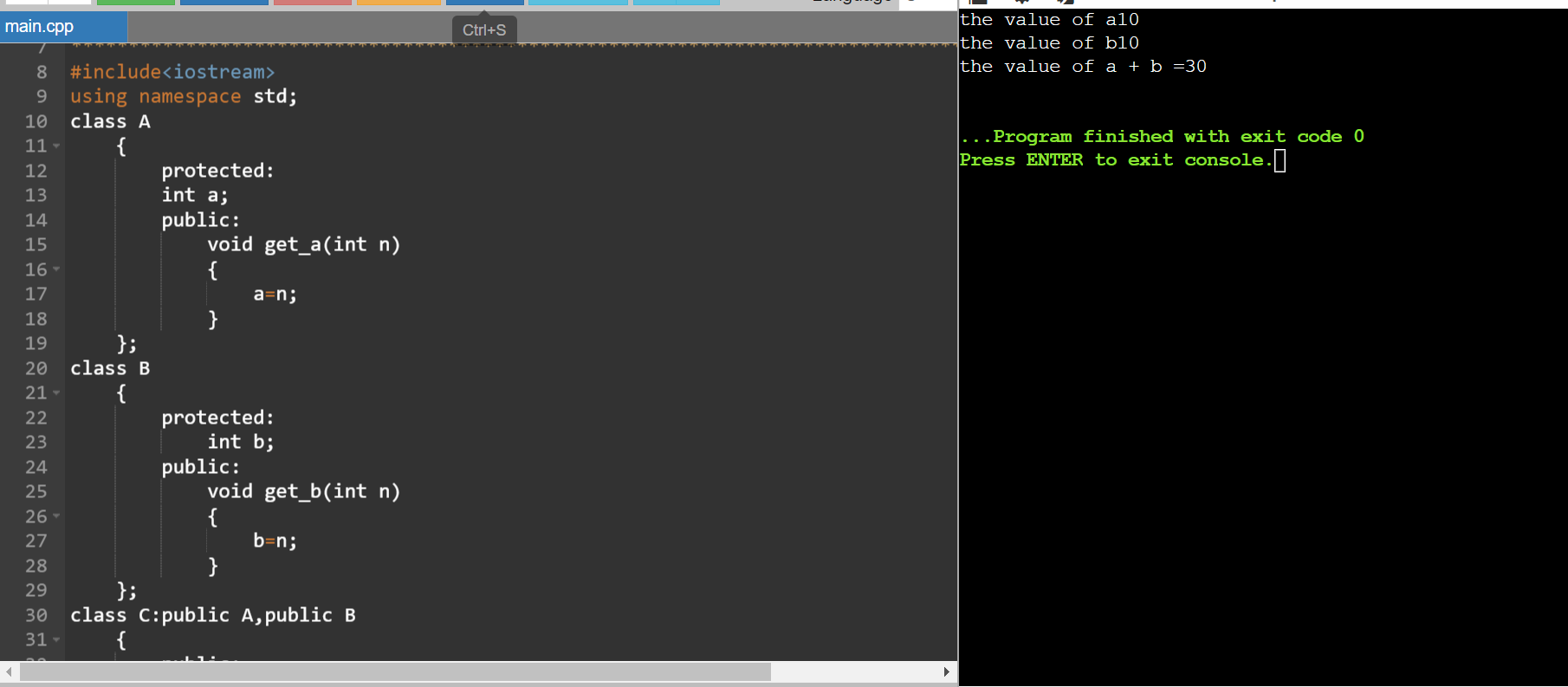
Program 1:single inheritance



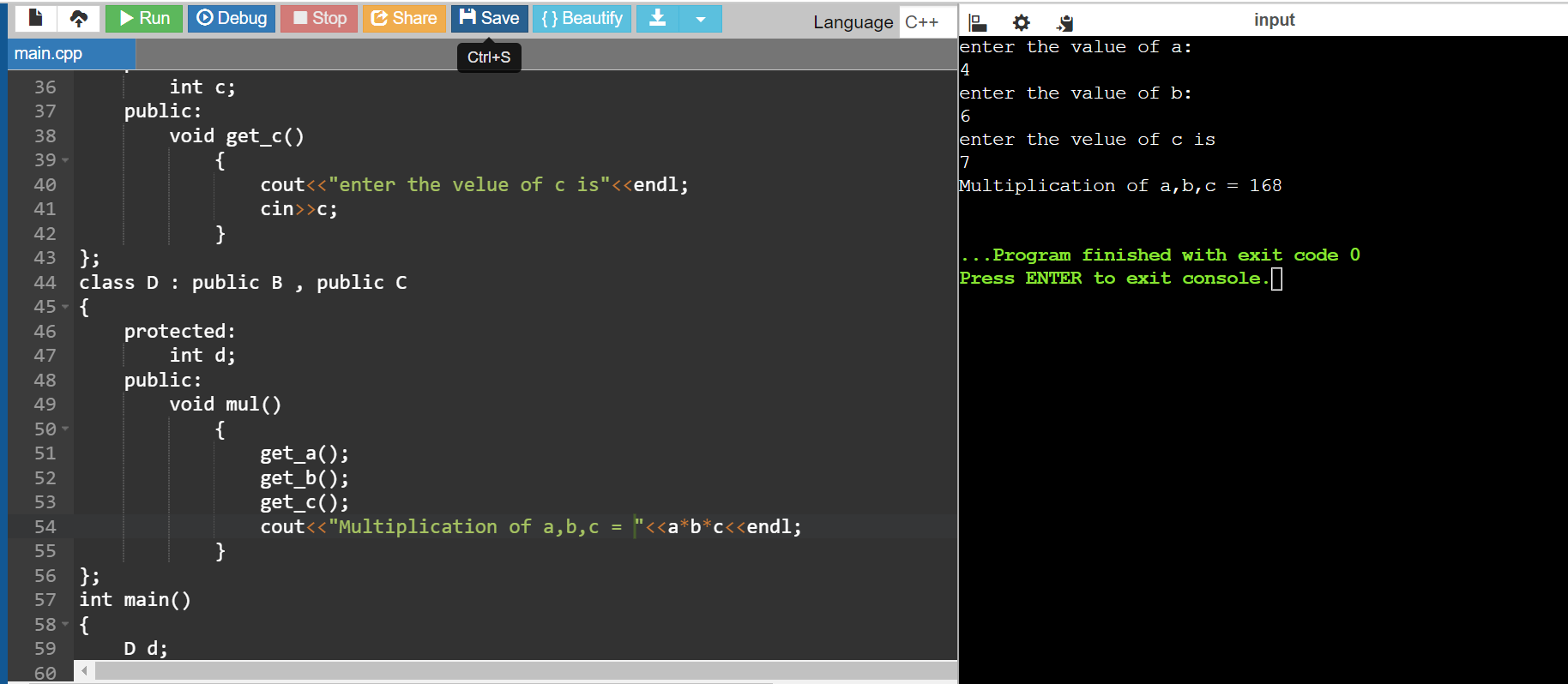
Program 2:



Program 3:



Program 4:



Program 5:

#include <iostream>

#include <string>

#include <vector>

using namespace std;

class Person {

protected:

string name;

int age;

public:

// Constructor

Person(string n, int a) : name(n), age(a) {}

// Virtual function to print person details

virtual void getDetails() const {

cout << "Name: " << name << "\nAge: " << age << endl;

}

virtual ~Person() {}

};

class Student : public Person {

private:

int studentId;

string major;

public:

Student(string n, int a, int id, string m) : Person(n, a), studentId(id), major(m) {}

void setMajor(string m) {

major = m;

}

string getMajor() const {

return major;

}

void getDetails() const override {

Person::getDetails();

cout << "Student ID: " << studentId << "\nMajor: " << major << endl;

}

};

class Faculty : public Person {

private:

string department;

int employeeId;

public:

Faculty(string n, int a, int id, string dept) : Person(n, a), employeeId(id), department(dept) {}

void setDepartment(string dept) {

department = dept;

}

string getDepartment() const {

return department;

}

void getDetails() const override {

Person::getDetails();

cout << "Employee ID: " << employeeId << "\nDepartment: " << department << endl;

}

};

class TeachingAssistant : public Student {

private:

vector<string> coursesTeaching;

public:

TeachingAssistant(string n, int a, int id, string m, vector<string> courses)

: Student(n, a, id, m), coursesTeaching(courses) {}

void setCoursesTeaching(vector<string> courses) {

coursesTeaching = courses;

}

vector<string> getCoursesTeaching() const {

return coursesTeaching;

}

void getDetails() const override {

Student::getDetails();

cout << "Courses Teaching: ";

for (const auto& course : coursesTeaching) {

cout << course << " ";

}

cout << endl;

}

};

class ResearchAssistant : public Person {

private:

string researchArea;

string supervisor;

public:

ResearchAssistant(string n, int a, string area, string sup)

: Person(n, a), researchArea(area), supervisor(sup) {}

void setResearchArea(string area) {

researchArea = area;

}

string getResearchArea() const {

return researchArea;

}

void setSupervisor(string sup) {

supervisor = sup;

}

string getSupervisor() const {

return supervisor;

}

void getDetails() const override {

Person::getDetails();

cout << "Research Area: " << researchArea << "\nSupervisor: " << supervisor << endl;

}

};

class GraduateStudentTA : public TeachingAssistant {

public:

GraduateStudentTA(string n, int a, int id, string m, vector<string> courses)

: TeachingAssistant(n, a, id, m, courses) {}

void getDetails() const override {

TeachingAssistant::getDetails();

}

};

int main() {

Student student("Ram", 21, 12345, "Computer Science");

student.getDetails();

cout << endl;

Faculty faculty("shree", 23, 67890, "Mathematics");

faculty.getDetails();

cout << endl;

vector<string> courses = {"CS101", "CS102"};

TeachingAssistant ta("Bhim", 29, 54321, "Physics", courses);

ta.getDetails();

cout << endl;

ResearchAssistant ra("ved", 36, "Machine Learning", "Dr. Smith");

ra.getDetails();

cout << endl;

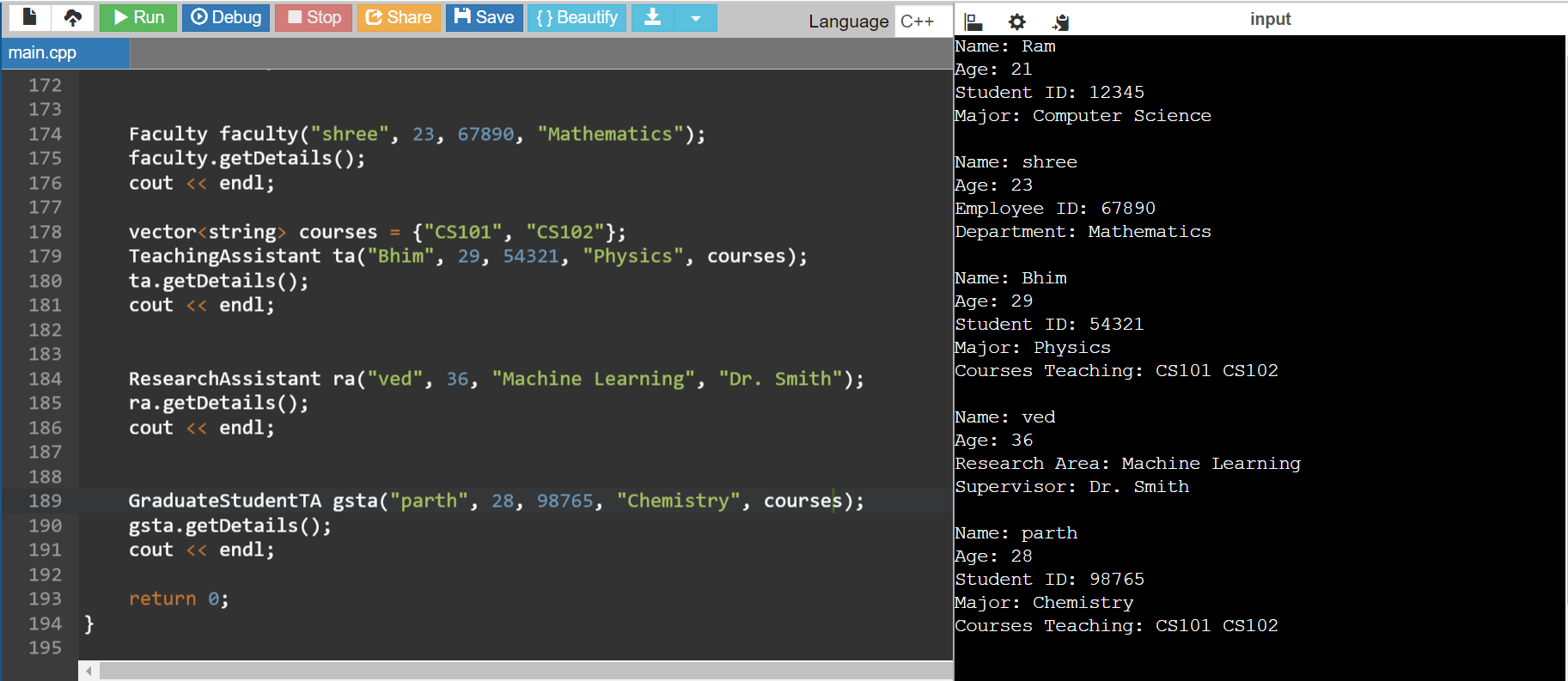
GraduateStudentTA gsta("parth", 28, 98765, "Chemistry", courses);

gsta.getDetails();

cout << endl;

return 0;

}



Program 6: Accesses Specifier

#include <iostream>

#include <string>

#include <vector>

#include <numeric>

using namespace std;

class Person {

protected:

string name;

int id;

public:

Person(string name, int id) : name(name), id(id) {}

virtual void getDetails() {

cout << "Name: " << name << ", ID: " << id << endl;

}

virtual ~Person() {}

};

class Student : public Person {

private:

string major;

double gpa;

public:

Student(string name, int id, string major, double gpa)

: Person(name, id), major(major), gpa(gpa) {}

void setMajor(string newMajor) {

major = newMajor;

}

string getMajor() const {

return major;

}

double calculateSemesterGPA(vector<double> grades) {

if (grades.empty()) return 0.0;

double sum = accumulate(grades.begin(), grades.end(), 0.0);

return sum / grades.size();

}

void getDetails() override {

Person::getDetails();

cout << "Major: " << major << ", GPA: " << gpa << endl;

}

};

class Faculty : public Person {

private:

string department;

string title;

public:

Faculty(string name, int id, string department, string title)

: Person(name, id), department(department), title(title) {}

void setDepartment(string newDepartment) {

department = newDepartment;

}

string getDepartment() const {

return department;

}

void teachCourse(string courseName) {

cout << title << " " << name << " is now teaching " << courseName << endl;

}

void getDetails() override {

Person::getDetails();

cout << "Department: " << department << ", Title: " << title << endl;

}

};

int main() {

Student s1("krishna", 1001, "Computer Science", 4.8);

s1.getDetails();

vector<double> grades = {3.5, 3.7, 4.0, 3.8};

double semesterGPA = s1.calculateSemesterGPA(grades);

cout << "Semester GPA: " << semesterGPA << endl;

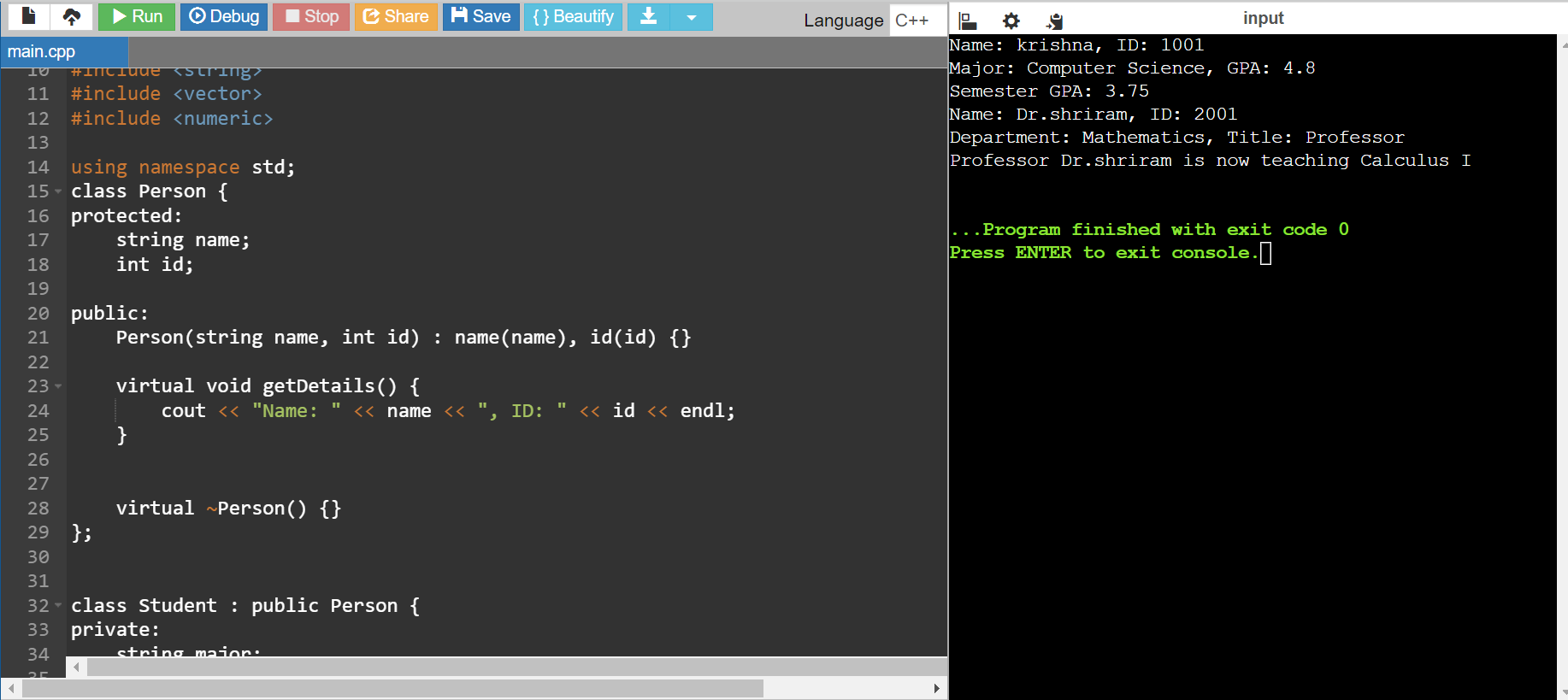
Faculty f1("Dr.shriram", 2001, "Mathematics", "Professor");

f1.getDetails();

f1.teachCourse("Calculus I");

return 0;

}



Program 7:Function Overloading

#include <iostream>

using namespace std;

class cal

{

public:

static int add(int a,int b)

{

return a+b;

}

static int add(int a,int b,int c)

{

return a+b+c;

}

static int sub(int a,int b)

{

return a-b;

}

static int sub(int a,int b,int c)

{

return a-b-c;

}

static int mul(int a,int b)

{

return a\*b;

}

static int divi(int a,int b)

{

return a/b;

}

};

int main()

{

cal C;

cout<<C.add(10,20)<<endl;

cout<<C.add(12,20,23)<<endl;

cout<<C.sub(30,20)<<endl;

cout<<C.sub(12,20,40)<<endl;

cout<<C.mul(12,20)<<endl;

cout<<C.divi(5,10)<<endl;

return 0;

}

