**Marwa Jabbar**

**BSCS 2A**

**SAP # 47129**

**OOP**

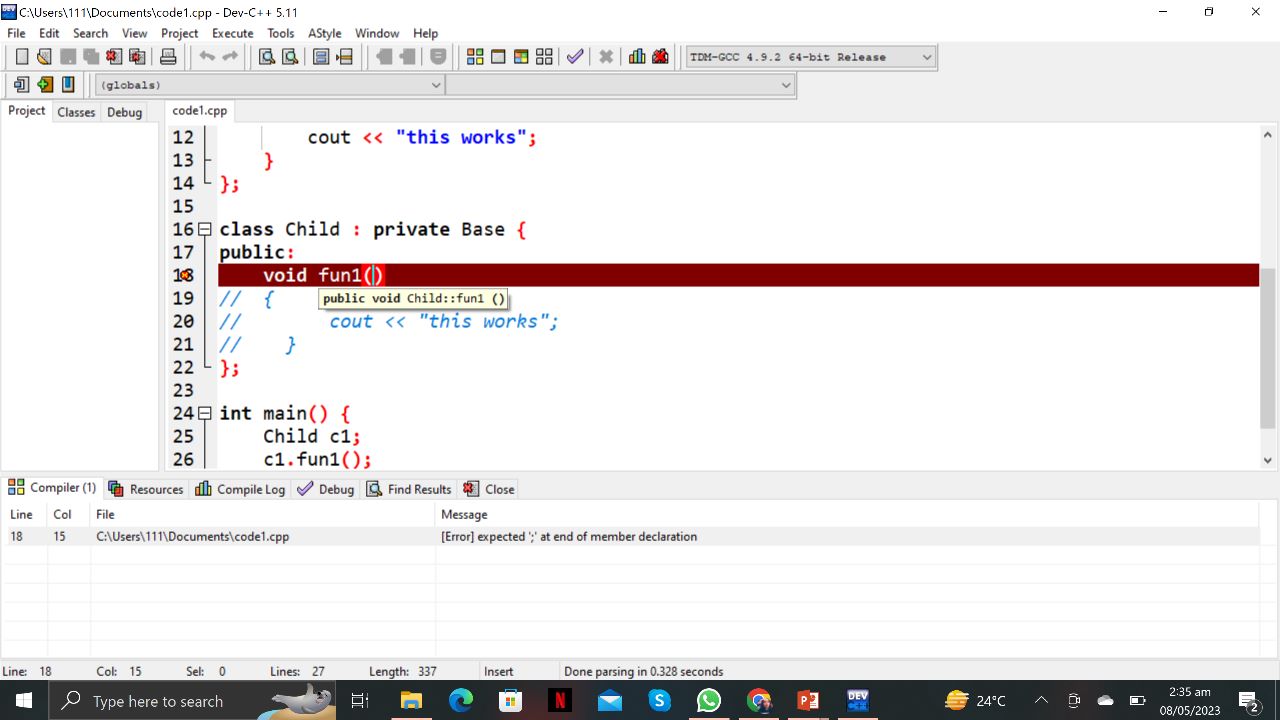
**QUESTION # 1:**

**Write the outputs of following code snippets.**

**PART A:**

**ANSWER;**

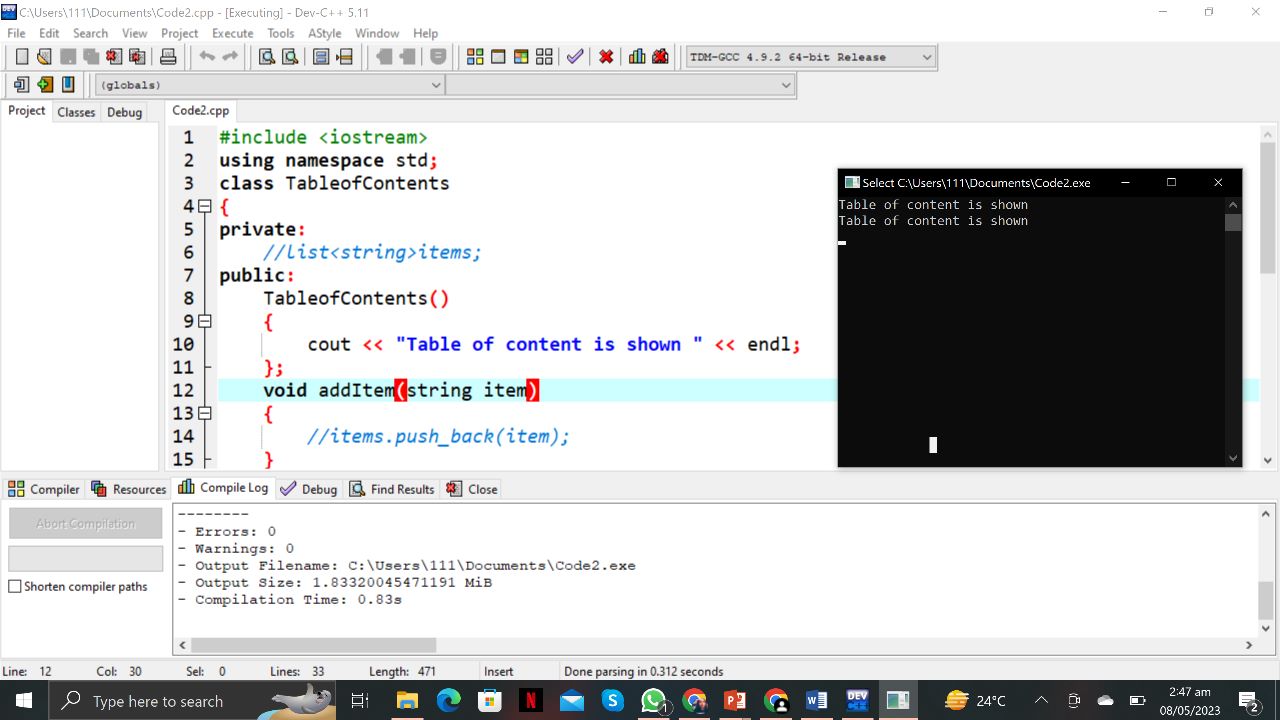
**ERROR**

****

**Part b:**

**ANSWER:**

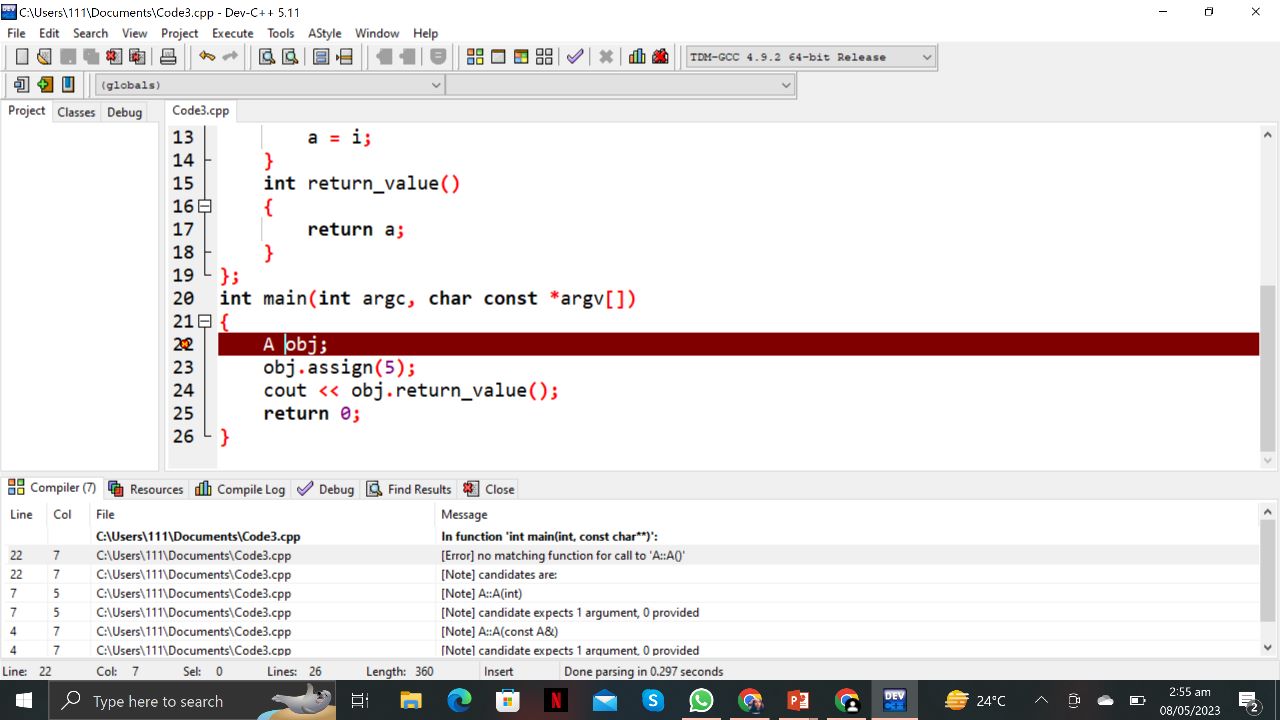
**TABLE OF THE CONTENT**

****

**PART C:**

**ANSWER:**

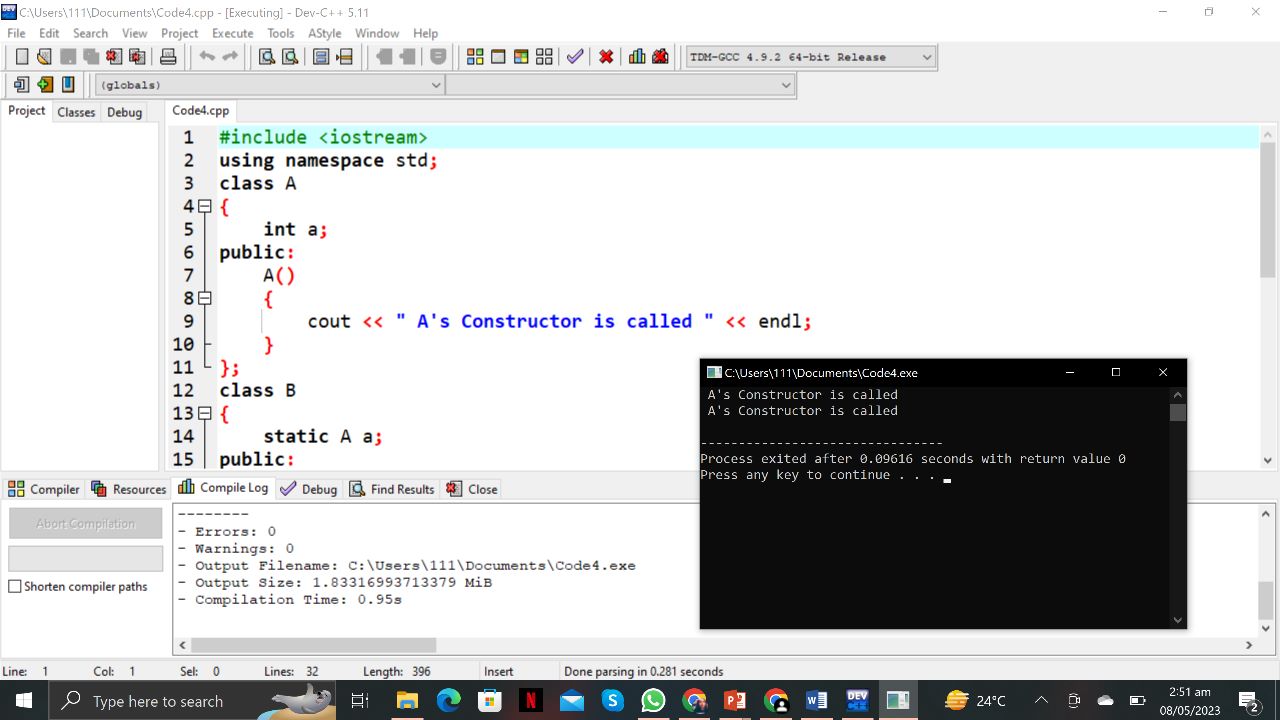
**Error**

****

**PART D**

**ANSWER:**

**A’S CONSTRUCTOR IS CALLED**

****

**QUESTION NO 2:**

1. **What is the purpose of access modifiers in OOP languages?**

**Answer.** The purpose of access modifiers in OOP languages is to change or modify the access of data members or member functions in a class.

**B) If we want to access the private members of a class in the child class what do we need to change?**

**Answer.** Private members of a class are not visible outside the class, including in child classes. However, there are certain cases where you might want to access private members of a class in a child class. In order to achieve this, you need to use the concept of inheritance and access modifiers.

To access private members of a parent class in a child class, you can change the access modifier of those members to protected. The protected access modifier allows the members to be accessed by the child class, as well as any other class that extends the parent class.

**C)Determine the accessibility of functions and data members in the following scenarios.**

|  |  |  |
| --- | --- | --- |
| **Scenario** | **Accessible** | **Not Accessible** |
| A private data member is declared in a class in accessibility by its object in the main function. |  | Yes |
| A protected function defined in parent class by the functions of the child class. | Yes |  |
| A public data of the parent class by the object of child class. | Yes |  |

**QUESTION 3:**

#include <iostream>

using namespace std;

class B1

{

int i;

int j;

void g()

{

cout << "---HELLO---";

}

};

class B2 : public B1

{

public:

int j;

void g()

{

cout << "---HELLO---";

}

};

class D : public B2

{

public:

int i;

};

int main()

{

D o;

D \*dptr = &o;

dptr ->i = 5;

dptr->j = 10;

o.g();

}

**QUESTION 4:**

**CODE :**

#include <iostream>

#include <string>

#include <vector>

using namespace std;

// Base class for all characters

class Character {

protected:

int id;

string name;

int max\_power;

int strength;

public:

Character(int id, string name, int max\_power, int strength) {

this->id = id;

this->name = name;

this->max\_power = max\_power;

this->strength = strength;

}

virtual void walk() {

cout << name << " is walking." << endl;

}

virtual void jump() {

cout << name << " is jumping." << endl;

}

virtual void eat() {

cout << name << " is eating." << endl;

}

};

// Derived class for Doremon

class Doremon : public Character {

private:

vector<string> gadgets;

string partner\_name;

public:

Doremon(int id, string name, int max\_power, int strength, vector<string> gadgets, string partner\_name)

: Character(id, name, max\_power, strength) {

this->gadgets = gadgets;

this->partner\_name = partner\_name;

}

void show\_gadgets() {

cout << "Gadgets:" << endl;

for (auto gadget : gadgets) {

cout << "- " << gadget << endl;

}

}

void launch\_attack() {

cout << name << " is launching an attack." << endl;

}

void fly() {

cout << name << " is flying." << endl;

}

};

// Derived class for Benten

class Benten : public Character {

private:

string watch\_name;

vector<string> powers;

int watch\_charge;

public:

Benten(int id, string name, int max\_power, int strength, string watch\_name, vector<string> powers, int watch\_charge)

: Character(id, name, max\_power, strength) {

this->watch\_name = watch\_name;

this->powers = powers;

this->watch\_charge = watch\_charge;

}

void rotate\_watch() {

cout << name << " is rotating the watch." << endl;

}

void fight() {

cout << name << " is fighting." << endl;

}

void drive() {

cout << name << " is driving." << endl;

}

};

// Main function to test the game

int main() {

// Create a Doremon character

vector<string> doremon\_gadgets = {"Bamboo Copter", "Anywhere Door", "Small Light"};

Doremon doremon(1, "Doremon", 100, 80, doremon\_gadgets, "Nobita");

// Create a Benten character

vector<string> benten\_powers = {"Water", "Fire", "Wind"};

Benten benten(2, "Benten", 120, 90, "Omnitrix", benten\_powers, 50);

// Test common actions

doremon.walk();

benten.jump();

doremon.eat();

// Test Doremon's unique actions

doremon.show\_gadgets();

doremon.launch\_attack();

doremon.fly();

// Test Benten's unique actions

benten.rotate\_watch();

benten.fight();

benten.drive();

return 0;

}