LAB 7 (SECJ1023)

PROGRAMMING TECHNIQUE II

SECTION 04 & 05, SEM 2, 2024/2025

1. Based on the Program 7.1, answer the following questions.

```
//Program 7.1
class Shape{
      private:
              double area;
      public:
             void setArea(double);
             double getArea();
};

void Shape::setArea(double area){
     this->area=area;
}

double Shape::getArea(){
       return area;
}

class Circle:public Shape{
      private:
              double radius;
      public:
             void setRadius(double);
             double getRadius();
};

void Circle::setRadius(double radius){
     this->radius=radius;
}

double Circle::getRadius(){
       return radius;
}
```

a. Draw a UML class diagram to show the relationship between these two classes.

b. When a `Circle` object is created, what are its public members?

c. What members of the `Shape` class are not accessible to the `Circle` class's methods?

d. Assuming a `main` method has the following declarations:

```
Shape s;
Circle c;
```

Indicate whether the following statements are legal or illegal:

```
c.setRadius(10.0);
s.setRadius(10.0);
cout<<c.getArea();
```

```
        cout<<s.getArea();
```

2. A superclass `Rectangle`, subclass `Square` and `main` method are shown in Program 7.2. You are required to fill in the blanks with the correct instruction according to the numbered questions.

```
//Program 7.2
#include <iostream>
using namespace std;

class Rectangle {
    protected:
       double width;
       double length;
       double calculateArea();
    public:
       Rectangle(double, double);
       void display();
};

Rectangle::Rectangle(double width, double length){
    _____(a)_____;
    _____(a)_____;
}

void Rectangle::display(){
    cout<<width<<endl;
    cout<<length<<endl;
}

double Rectangle::calculateArea(){
    return width*length;
}

class Square_____(b)_____{
    private:
       double height;
    public:
       _____(c)_____;
       double calculateVolume();
       void display();
};

_____(d)_____{
    this->height=height;
    this->width=width;
    this->length=length;
}

double Square::calculateVolume(){
    return _____(e)_____;
}

```

```
void Square::display(){
    _____(f)_____;
    cout<<height;
    cout<<calculateVolume();
}

int main(){
    _____(g)_____;
    _____(h)_____;
    _____(i)_____;
    return 0;
}
```

a. Assign values for superclass attributes.

b. Subclass `Square` inherits from superclass `Rectangle`.

c. A function prototype for `Square` constructor that inherits `Rectangle` constructor.

d. A respective function header for `Square` constructor based on (c).

e. Calculate the volume for `Square` by using the inherited method.

f. Display the width and length of `Square` by using the inherited method.

g. Create a `Square` object named `squareBox` and pass the values of `15`, `10`, `10` for `height`, `width` and `length` respectively.

h. Calculate the volume of `squareBox`.

i. Display the properties of `squareBox`.

3. Examine the class in Program 7.3. Table 7.1 lists the variables that are members of the `Words` class (some are inherited). Complete the table by filling in the access specification each member will have in the `Words` class. Write "inaccessible" if a member is inaccessible to the `Words` class.

```
//Program 7.3
class Book  {
   private:
      string name;
   protected:
      int page;
   public:
      string publisher;
};

class Dictionary:protected Book
{
   private:
      int numWords;
   protected:
      string meaning:
   public:
      string words;
};

class Words:public Dictionary
{
   private:
      int num;
   protected:
      string perkataan;
   public:
      string source;
};
```

Table 7.1: Access Specification for Member Variables in `Words` Class

Member Variable	Access Specification in `Words` Class
name	
page	
publisher	
numWords	
meaning	
words	
perkataan	
num	
source	

4. Given three classes named `Automobile`, `Car` and `Truck` respectively, and some objects of these classes as shown in Program 7.4, answer questions (a) to (c).

```
//Program 7.4
#include <iomanip>
#include <iostream>
using namespace std;

class Automobile
{
  private:
    string make;

  protected:
    int model;

  public:
    double price;

    Automobile()
    {
      make = "";
      model = 0;
      price = 0.0;
      cout << "An Automobile object has been created but "
           << "not yet have details." << endl;
```

```cpp
    }

    Automobile(string a, int b, double c)
    {
      make = a;
      model = b;
      price = c;
      cout << "Automobile object: " << make << " makes in "
           << model << ". The price is RM " << price << "."
           << endl;
    }

    void printInfo(){ }
}; //End class Automobile

class Car: private Automobile
{
  private:
    int doors;

  public:
    Car(): Automobile("BMW", 2010, 150000.0)
    {
      doors = 0;
      cout << "Car with car's make, year model and price "
           << "(\"BMW\", 2010, 150000.0) accordingly."
           << endl;
    }

    Car(string a, int b, double c, int d): Automobile(a, b, c)
    {
      doors = d;
      cout << a << "'s car has " << d << " doors." << endl;
    }

    void printCar() { }
}; //End class Car

class Truck: protected Car
{
  protected:
    string driveType;

  public:
    Truck(string d): Car("Toyota", 2014, 45000.0, 4)
    {
      driveType = d;
      cout << "This is the truck with " << d
           << " drive type." << endl;
    }

    void printTruck() { }
}; //End class Truck
```

```
78
79  int main()
80  {
81      cout << fixed << setprecision(2);
82
83      Car car;
84      Truck truck("4WD");
85      Automobile automobile;
86
87      return 0;
88  }
```

a. What will the lines 83 to 85 print onto the screen when the program runs?

   i. `Car car;`

   ii. `Truck truck("4WD");`

   iii. `Automobile automobile;`

b. Based on the objects created in the `main` function of the, determine the member variables that each object owns. Write (**Own**) in the corresponding blank cells, if the object owns or has a copy of the variable; and write (**No**) if otherwise.

	Member Variables				
Object	**`make`**	**`model`**	**`price`**	**`doors`**	**`driveType`**
`car`					
`truck`					
`automobile`	■	■	■		

c. With reference to the same member variables in (a), determine whether they are accessible inside the methods and by the object listed below. Write (**Accessible**) in the corresponding blank cells, if it is accessible; and write (**No**) if otherwise.

	Member Variables				
Object	**`make`**	**`model`**	**`price`**	**`doors`**	**`driveType`**
`printInfo()`		■	■		
`printCar()`			■	■	
`printTruck()`	■				
`automobile`					■

5. Compile and run Program 7.5. Understand what each line of the statement in the program does. Based on the program, draw the class diagram. In your own words, explain why such output is printed.

```
//Program 7.5
#include <iostream>
using namespace std;
class Transport  {
    public:
       Transport();
       Transport(const Transport&);
};

Transport::Transport(){
    cout<<"Transport default constructor is called"<<endl;
}

Transport::Transport(const Transport& trans){
    cout<<"Transport copy constructor is called"<<endl;
}

class Car: public Transport  {
    public:
       Car(){
          cout<<"Car default constructor is called"<<endl;
       }

       Car(const Car& car){
          cout<<"Car copy constructor is called"<<endl;
       }
};

class Motor: public Transport  {
    public:
       Motor(){
          cout<<"Motor default contructor is called"<<endl;
       }

       Motor(const Motor& motor){
          cout<<"Motor copy constructor is called"<<endl;
       }
};

int main()  {
    Car car1;
    Motor motor1;
    Transport trans = car1;
    return 0;
}
```

6. Compile and run Program 7.6. Understand what each line of the statement in the program does. Based on the program, draw the class diagram. In your own words, explain why such output is printed.

```
//Program 7.6
#include <iostream>
using namespace std;
class Thought{
    public:
       void message(){
          cout<<"Some people get lost in thought"<<endl;
       }
};

class Advice: public Thought{
    public:
       void message(){
          cout<<"Avoid cliches like the plague"<<endl;
       }
};

int main(){
    Thought think;
    Advice cliche;
    think.message();
    cliche.message();
    return 0;
}
```

Now, run the following Program 7.7. Why such output is printed?

```
//Program 7.7
#include <iostream>
using namespace std;
class Thought  {
    public:
       void message()  {
          cout<<"Some people get lost in thought"<<endl;
       }
};

class Advice:public Thought  {
    public:
       void message()  {
          Thought::message();
          cout<<"Avoid cliches like the plague"<<endl;
       }
};

int main()  {
    Thought think;
    Advice cliche;
    think.message();
    cliche.message();
    return 0;
}
```

7. Based on Program 7.8:

   a. Identify which `showValue` function applies the overriding and overloading concept.

   b. Trace the output of the program.

   c. Define the difference between method overriding and method overloading.

```
//Program 7.8
#include <iostream>
using namespace std;

class Superclass3{
    public:
       void showValue(int a){
          cout<<a<<endl;
       }
};

class Subclass3:public Superclass3{
    public:
       void showValue(int a){
          cout<<"The subclass: \t"<<a<<endl;
       }

       void showValue(double a){
          cout<<a<<endl;
       }
};

int main(){
    Subclass3 obj;
    obj.showValue(8);
    obj.showValue(8.97);
    return 0;
}
```

8. Consider the following two classes in Program 7.9.

```
//Program 7.9
#include<iostream>
using namespace std;

class B{
    private:
       int k;

    protected:
       int i;
       int j;
       void printMe();

    public:
       B(int j){
           this->j=j;
       }

       void changeValue(int);
};

void B::printMe(){
```

```
23      cout<<"i:"<<i<<" j:"<<j<<" k:"<<k<<endl;
24  }
25  
26  void B::changeValue(int j){
27      this->j = j;
28  }
29  
30  class C:public B{
31      private:
32         int j;
33  
34      public:
35         C(int value){
36             i = 6;
37             j = 10;
38         }
39         void printMe();
40  
41      protected:
42         void changeValue(int);
43  };
44  
45  void C::printMe(){
46      B::printMe();
47      cout<<"j: "<<j;
48  }
49  
50  void C::changeValue(int j){
51      B::changeValue(j);
52      this->i=25;
53  }
54  
55  int main(){
56      C obj(12);
57      obj.changeValue(45);
58      obj.printMe();
59  }
```

The Program 7.9 will result in a compilation error. You are required to:

a. Identify which line(s) of code result in s compilation error(s).

b. Give a reason(s) for the error(s).

c. Correct the statement(s).

Write the output of the program once the error(s) have been corrected.

9. Given Program 7.10 that uses multiple inheritance:

```
1  //Program 7.10
2  #include <string>
3  #include <iostream>
4  using namespace std;
```

```cpp

class Person
{
    private:
       string name;
       int age;
       bool IsMale;

    public:
        Person(string name,int  age,bool bIsMale:
        name(name),age(age),IsMale(bIsMale){}

        string getName() { return name; }
        int getAge() { return age; }
        bool getIsMale() { return IsMale; }
};

class Employee
{
    private:
       string m_strEmployer;
       double m_dWage;

    public:
        Employee(string strEmployer, double dWage):
        m_strEmployer(strEmployer), m_dWage(dWage){}

        string GetEmployer() { return m_strEmployer; }
        double GetWage() { return m_dWage; }
};

class Teacher: public Person, public Employee
{
    private:
       int m_nTeachesGrade;

    public:
        Teacher(std::string strName, int nAge, bool bIsMale,
        string strEmployer, double dWage, int nTeachesGrade):
        Person(strName, nAge, bIsMale), Employee(strEmployer,
        dWage), m_nTeachesGrade(nTeachesGrade){}

        void display(){
           cout<<"Name:\t"<<Person::getName()<<endl;
           cout<<"Age:\t"<<Person::getAge()<<endl;
           cout<<"Employer:\t"<<Employee::GetEmployer()
               <<endl;
           cout<<"Wage:\t"<<Employee::GetWage()<<endl;
           cout<<"Grade:\t"<<m_nTeachesGrade<<endl;}
};

int main(){
    Teacher teach1 ("Allan", 35, true, "UTM", 4500, 45);
    teach1.display();
    return 0;
}
```

a. Save Program 7.10 as `PersonRelation.cpp`. Then compile and run Program 7.10. Understand what each line of the statement in the program does. Based on the program, draw the class diagram. In your own words, explain why such output is printed.

b. Now, remove the `Person` and `Employee` classes from the Program 7.10 and place the classes in their own files. Use `#include` at the top of the `PersonRelation.cpp` file to include the two parent classes (eg.: `#include “person.cpp”`). Compile and run the program. You will get the same output as Program 7.10.

10. Create a class **`B`** that has the following:

a. `m`, an integer. This member variable should not be accessible to code outside the class or to any its subclass.

b. `n`, an integer. This member variable should be accessible only to its subclass.

c. `setM, getM, setN` and `getN`. These are the mutator and accessor methods for the member variables `m` and `n`. These methods should be accessible to code outside the class.

d. `calc`. A public method that returns the value of `m` multiplies `n`.

e. Create a class `D` that is a subclass to `B`. The class members are:

i. `q`, a double. This member variable should not be accessible to code outside the class.

ii. `r`, a double. This member variable should be accessible only to its subclass.

iii. `setQ, setR, getQ` and `getR`. These are the mutator and accessor methods for the member variable `q` and `r`. These methods should be accessible to code outside the class.

iv. `calc`. A public method that overrides the superclass `calc` method and return the value of `q` multiplies `r`.

f. Create a `main` method to test the functionality of your `B` and `D` classes.

11. Examine the following class declarations in Program 7.11:

```
//Program 7.11
class Faculty
{  protected:
      string facName;
      string buildingNo;

   public:
      Faculty (string,string);
};
```

```

Faculty::Faculty(string facName, string buildingNo){
   this->facName=facName;
   this->buildingNo=buildingNo;
}

class Student
{  protected:
      string name;

   public:
      Student(string name)
      { this->name=name;}
};
```

Write the declaration of a class named `TeachingAssistant`. The class should be derived from both the `Faculty` and `Student` classes above. This should be a case of multiple inheritance, where both `Faculty` and `Student` are base classes. Draw its complete class diagram.

a. The `TeachingAssistant` class will have an attribute `course` of type string.

b. The `TeachingAssistant` class will also call constructors of its base classes.

12. Based on the following UML class diagram in Figure 7.1, write a complete C++ program that implements the relationships among classes. The explanation of each method is as in Table 7.2.

Table 7.2: Explanation of each method in each class

Methods	**Explanation**
`Flight class`	
`addPassenger(Passenger)`	This method will add `Passenger` objects to the `passengerList`.

`printInfo()`	This method will display all flight information as an example below:
	`Flight No:  MH3120`
	`Destination: Johor Bahru`
	`Departure: 8:10`
	`Arrival: 9:00`
	`Number of Passengers: 10`

`Passenger`	
`printDetails()`	This method will display all passenger information, i.e. `name` and `age`.
`Kids class`	
`printDetails()`	• This method will display all kids passenger information. • This method overrides its superclass `printDetails` method. The values of its

	inherited attributes will be displayed by calling its inherited method.
`Adult class`	
`printDetails()`	• This method will display all adults passenger information. • This method overrides its superclass `printDetails` method. The values of its inherited attributes will be displayed by calling its inherited method.
`Time class`	
`getHour()`	This method will return the attribute value of `hour`.
`getMinute()`	This method will return the attribute value of `minute`.

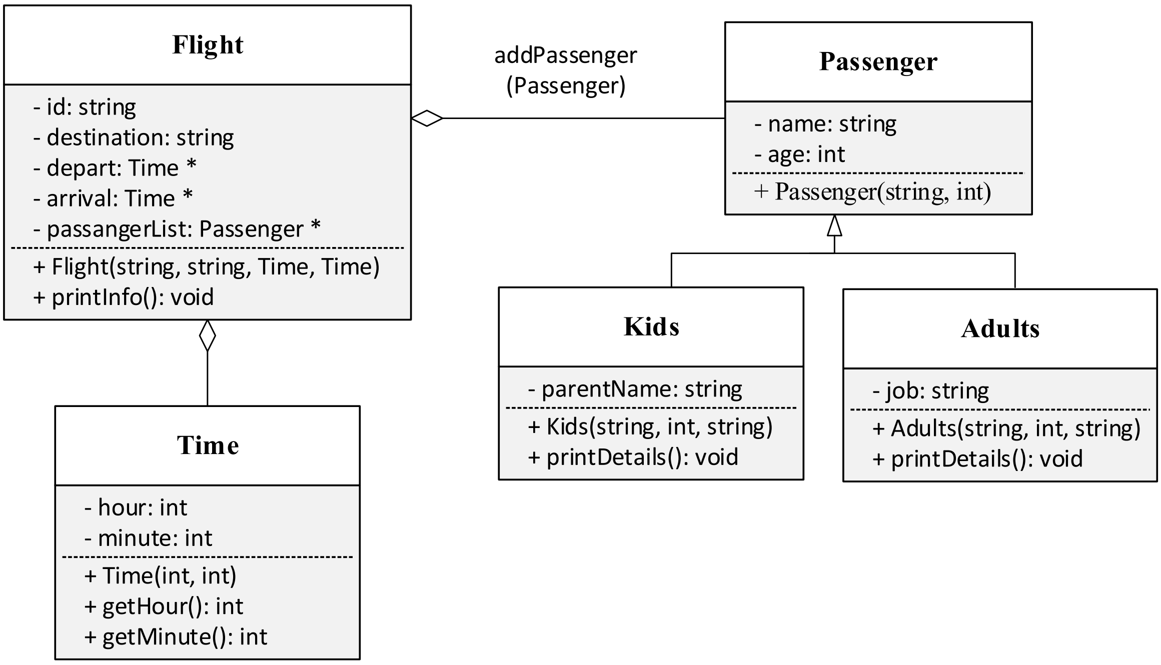

Figure 7.1: Class Diagram of `Flight`, `Passenger`, `Kids`, `Adult` and `Time` classes

Your program must be able to produce the following output as illustrated in Figure 7.2.

```
Flight Number:  NAS7921
Destination:    Johor Bahru
Departure:      8:10
Arrival:        9:00

Number of Passengers: 3
Number of Adults:     2
Number of Kids:       1

Passengers Details:
Name:    Ali      Age:    34      Job Title        Lecturer
Name:    Goh      Age:    25      Job Title        Student
Name:    Azmi     Age:    3       Parents Name:    Ali
```

Figure 7.2: Example of expected output

13. Faculty of Computing will be organizing a competition that is open to all UTM students. The competition is meant to provide a platform to the students to exhibit their idea. Some important notes regarding the competition:

    a. It is an individual participation.

    b. Each student must have a project beforehand.

    c. Each student must find an advisor which can be any UTM staff regardless of their position. That means, besides lecturers, technicians, research officers, etc, also can be appointed as advisers. However, only one advisor is allowed for each student.

    d. Also, each staff can only advise a student.

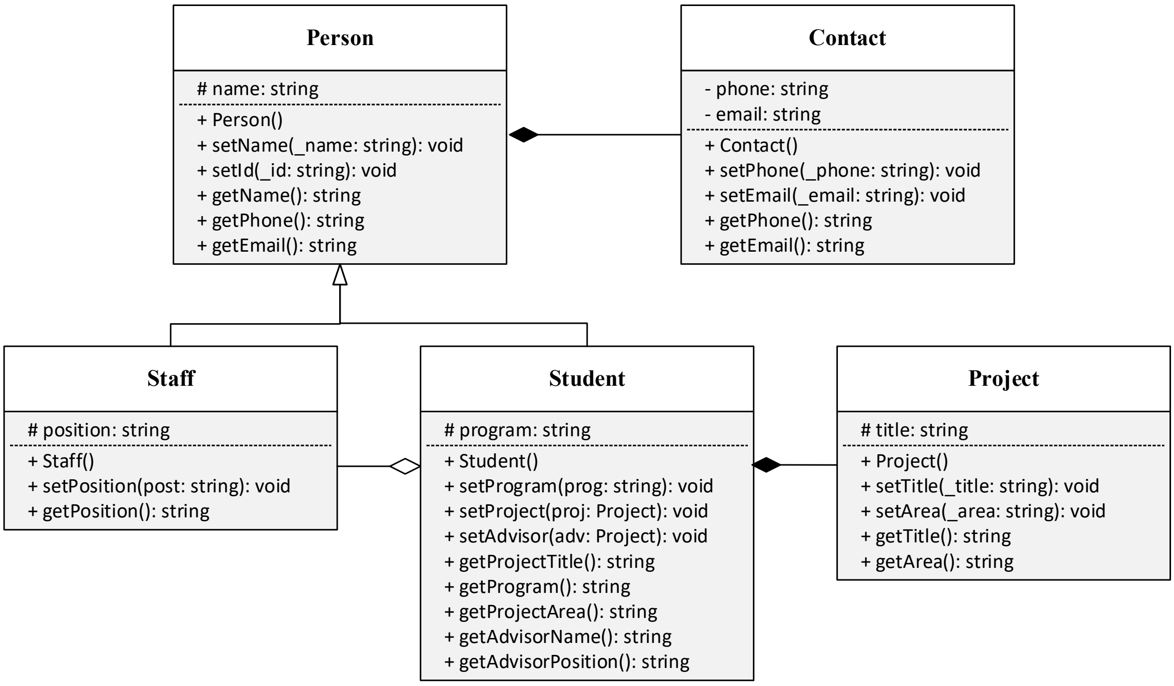
Person
# name: string
+ Person()
+ setName(_name: string): void
+ setId(_id: string): void
+ getName(): string
+ getPhone(): string
+ getEmail(): string
Contact
- phone: string
- email: string
+ Contact()
+ setPhone(_phone: string): void
+ setEmail(_email: string): void
+ getPhone(): string
+ getEmail(): string
Staff
# position: string
+ Staff()
+ setPosition(post: string): void
+ getPosition(): string
Student
# program: string
+ Student()
+ setProgram(prog: string): void
+ setProject(proj: Project): void
+ setAdvisor(adv: Project): void
+ getProjectTitle(): string
+ getProgram(): string
+ getProjectArea(): string
+ getAdvisorName(): string
+ getAdvisorPosition(): string
Project
# title: string
+ Project()
+ setTitle(_title: string): void
+ setArea(_area: string): void
+ getTitle(): string
+ getArea(): string

Figure 7.3: UML class diagram

Suppose the class design has been done by other parties as shown in Figure 7.3. Based on the class diagram in Figure 7.3 and the requirements above, write a C++ program that does the following tasks:

a. Implement all the classes and their relationships. You need to define all the methods listed in the diagram, and you can add other methods whenever possible.

b. Create a list of `Advisor` objects and another list of `Student` objects. Use the sample data given in Table 7.3 and Table 7.4.

c. Assign each student with an advisor.

d. Print the information of all the students along with their projects and advisers.

e. Suppose the first and second students want to change their advisor each other, while the advisor for the third student wants to withdraw from being appointed as an advisor. Write the code to implement these situations.

f. Print again the final list.

Table 7.3: Sample data of advisors

No	Advisor's Name	Phone	Email	Position
1	Prof. Dr. Muhammad Roslan Abdullah	0199875678	mroslana29@gmail.com	Lecturer
2	Mr. Qamarool Zaman	01156781234	qam23@gmail.com	Senior Technician
3	Dr. Siti Zubaidah Rosli	0101117456	ctzr983@gmail.com	Research Officer

Table 7.4: Sample data of students

No	Student's Name	Phone	Email	Program	Project
1	Lim Sew Ying	010897634045	lsying12@live.utm.my	Electrical Engineering	Smart Dustbin
2	Abu Bakar Razali	0111210000	abraz78@live.utm.my	Computer Science	Money Recognition for Blind People
3	Nur Amalina Muhammad	0129000123	nam978@live.utm.my	Biomedical Engineering	Dengue Testing Kit