CSE Assignment (C++)

Problem 1

Question:

> *Create a class called InputData. It has two private data members data_a and data_b. Set thevalues of these two data members by using two public functions get_a() and get_b(). Derive a class called Arith_Unit from InputData. It contains the functions add(),sub(), mul(),div() to perform arithmetic operations on data_a and data_b. Derive a class Logic_Unit from InputData. It contains the functions and(), or() and xor() to perform logical operations on data_a and data_b. Finally derive a class called ALUnit from Arith_Unit and Logic_Unit classes. It has to perform arithmetic and logical operations according to the given codes. Choose code 0 to code 6 to perform the said seven operations. Write sample program to test the ALU class.*

Code:

```
#include <iostream>

using namespace std;

class InputData;
class Arith_Unit;
class Logic_Unit;
class ALUnit;

class InputData
{
private:
    int data_a;
    int data_b;

public:
    InputData();
    InputData(int a, int b);

    int get_a() const;
    int get_b() const;
};

class Arith_Unit : private InputData
{
public:
    Arith_Unit()
    {
    }

    Arith_Unit(int a, int b) : InputData(a, b)
    {
```

```
    }

    int add_a() const;
    int sub_a() const;
    int mul_a() const;
    int div_a() const;
};

class Logic_Unit : private InputData
{
public:
    Logic_Unit()
    {
    }

    Logic_Unit(int a, int b) : InputData(a, b)
    {
    }

    int and_l() const;
    int or_l() const;
    int xor_l() const;
};

class ALUnit : public Arith_Unit, public Logic_Unit
{
public:
    ALUnit()
    {
    }

    ALUnit(int a, int b) : Arith_Unit(a, b), Logic_Unit(a, b)
    {
    }
};

InputData::InputData()
{
    data_a = 0;
    data_b = 0;
}

InputData::InputData(int a, int b)
{
    data_a = a;
    data_b = b;
}

int InputData::get_a() const
{
    return data_a;
}
```

```
int InputData::get_b() const
{
    return data_b;
}

int Arith_Unit::add_a() const
{
    return get_a() + get_b();
}

int Arith_Unit::sub_a() const
{
    return get_a() - get_b();
}

int Arith_Unit::mul_a() const
{
    return get_a() * get_b();
}

int Arith_Unit::div_a() const
{
    return get_a() / get_b();
}

int Logic_Unit::and_l() const
{
    return get_a() & get_b();
}

int Logic_Unit::or_l() const
{
    return get_a() | get_b();
}

int Logic_Unit::xor_l() const
{
    return get_a() ^ get_b();
}

int main(int argc, char const *argv[])
{
    ALUnit test(2, 3);

    cout << test.add_a() << endl;
    cout << test.sub_a() << endl;
    cout << test.mul_a() << endl;
    cout << test.div_a() << endl;
    cout << test.and_l() << endl;
    cout << test.or_l() << endl;
    cout << test.xor_l() << endl;
```

```
    return 0;
}
```

Output:

```
6
0
2
3
1
```

Problem 2

Question:

> *Define a class Employee with data members as empno, name and designation. Derive a classQualification from Employee that has data members UG, PG and experience. Create another class Salary which is derived from both these classes to display the details of the employee and compute their increments based on their experience and qualifications*

Code:

```
#include <iostream>

using namespace std;

class Employee;
class Qualification;
class Salary;

class Employee
{
private:
    string empno;
    string name;
    string designation;

public:
    Employee();
    Employee(string eno, string peru, string desig);

    void setFields(string eno, string peru, string desig);
    string get(string field) const;
};

class Qualification : public Employee
{
private:
    bool UG;
```

```
    bool PG;
    int experience;

public:
    Qualification();
    Qualification(bool under, bool post, int exp);

    void setFields(bool under, bool post, int exp);
    int get(string field) const;
};

class Salary : public Qualification
{
private:
    int salary;

public:
    Salary();
    Salary(int income);

    int computeIncrement() const;
    void dispDetails() const;
};

Employee::Employee()
{
    empno = "";
    name = "";
    designation = "";
}

Employee::Employee(string eno, string peru, string desig)
{
    empno = eno;
    name = peru;
    designation = desig;
}

void Employee::setFields(string eno, string peru, string desig)
{
    empno = eno;
    name = peru;
    designation = desig;
}

string Employee::get(string field) const
{
    if (field == "empno")
    {
        return empno;
    }
    else if (field == "name")
```

```
    {
        return name;
    }
    else if (field == "designation")
    {
        return designation;
    }

    return "";
}

Qualification::Qualification()
{
    UG = 0;
    PG = 0;
    experience = 0;
}

Qualification::Qualification(bool under, bool post, int exp)
{
    UG = under;
    PG = post;
    experience = exp;
}

void Qualification::setFields(bool under, bool post, int exp)
{
    UG = under;
    PG = post;
    experience = exp;
}

int Qualification::get(string field) const
{
    if (field == "UG")
    {
        return UG;
    }
    else if (field == "PG")
    {
        return PG;
    }
    else if (field == "experience")
    {
        return experience;
    }

    return -1;
}

int Salary::computeIncrement() const
{
```

```
    return 50 * (get("experience") * (get("UG") ? 2 : 1) * (get("PG") ? 3 : 1));
}

void Salary::dispDetails() const
{
    cout << "Employee Number : " << Employee::get("empno") << endl;
    cout << "Employee Name : " << Employee::get("name") << endl;
    cout << "Designation : " << Employee::get("designation") << endl;
    cout << "UG Completion : " << (get("UG") ? "Yes" : "No") << endl;
    cout << "PG Completion : " << (get("PG") ? "Yes" : "No") << endl;
    cout << "Experience : " << get("experience") << endl;
    cout << "Salary : " << salary << endl;
    cout << "Increment : " << computeIncrement() << endl;
}

Salary::Salary()
{
    salary = 0;
}

Salary::Salary(int income)
{
    salary = income;
}

int main(int argc, char const *argv[])
{
    Salary emp(24000);

    emp.Employee::setFields("123", "TZ", "HR");
    emp.setFields(1, 1, 5);
    emp.dispDetails();

    return 0;
}
```

Output:

```
Employee Number : 123
Employee Name : TZ
Designation : HR
UG Completion : Yes
PG Completion : Yes
Experience : 5
Salary : 24000
Increment : 1500
```

Problem 3

Question:

> *Create a class named EB_amount. It has the data members units_used and bill. Use memberfunction to set unit_used. Upto 200 units 3 rupees per unit, 201 to 500, 4 rupees per unit and above 500 5.5 rupees per unit are allotted by EB. Calculate the bill amount and display the amount. Create another class Salary with basic, DA and HRA. Set basic by a member function. 104 percent of basic is assigned as DA and 10 percent is allotted as HRA. Display the total salary. Derive a class Budget contains income, tuition_fee, house_rent, saving, grocery, eb_bill as data members. Set the values and get the values of income and eb_bill from parent classes. Display the budget details*

Code:

```
#include <iostream>

using namespace std;

class EB_amount;
class Salary;
class Budget;

class EB_amount
{
private:
    double units_used;

protected:
    double bill;

public:
    EB_amount();
    EB_amount(double units);

    void calcBill();
    void displayBill() const;
    void setUnits(double units);
};

class Salary
{
private:
    double basic;
    double DA;
    double HRA;

protected:
    double total;

public:
    Salary();
    Salary(double base);

    void assignBaseScaled();
```

```
    void displaySalary() const;
    void setBase(double base);
};

class Budget : public EB_amount, public Salary
{
private:
    double *income;
    double tuition_fee;
    double house_rent;
    double saving;
    double *eb_bill;

public:
    Budget();
    Budget(double base, double tut_fee, double res_rent, double units_used);

    void updateSavings();
    void scaledVals(double base, double units_used);
    void displayBudget() const;
    void setField(string field, double value);
    void setAll(double base, double tut_fee, double res_rent, double units_used);
};

EB_amount::EB_amount()
{
    units_used = 0;
    bill = 0;
}

EB_amount::EB_amount(double units)
{
    units_used = units;
    bill = 0;

    calcBill();
}

void EB_amount::calcBill()
{
    if (units_used <= 200)
    {
        bill = units_used * 3;
    }
    else if (units_used > 200 && units_used <= 500)
    {
        bill = units_used * 4;
    }
    else if (units_used > 500)
    {
        bill = units_used * 5.5;
    }
```

```
    else
    {
        bill = -1;
    }
}

void EB_amount::displayBill() const
{
    cout << "EB Bill : " << bill;
}

void EB_amount::setUnits(double units)
{
    units_used = units;

    calcBill();
}

Salary::Salary()
{
    basic = 0;
    DA = 0;
    HRA = 0;
    total = 0;
}

Salary::Salary(double base)
{
    basic = base;

    assignBaseScaled();
}

void Salary::assignBaseScaled()
{
    DA = 1.04 * basic;
    HRA = 0.1 * basic;
    total = basic + DA + HRA;
}

void Salary::displaySalary() const
{
    cout << "Salary : " << total;
}

void Salary::setBase(double base)
{
    basic = base;

    assignBaseScaled();
}
```

```
Budget::Budget()
{
    income = &total;
    tuition_fee = 0;
    house_rent = 0;
    saving = 0;
    eb_bill = &bill;
}

Budget::Budget(double base, double tut_fee, double res_rent, double units_used)
{
    income = &total;
    tuition_fee = tut_fee;
    house_rent = res_rent;
    eb_bill = &bill;

    scaledVals(base, units_used);
}

void Budget::updateSavings()
{
    saving = *income - tuition_fee - house_rent - *eb_bill;
}

void Budget::scaledVals(double base, double units_used)
{
    setBase(base);
    setUnits(units_used);
    updateSavings();
}

void Budget::displayBudget() const
{
    cout << "Income : " << *income << endl;
    cout << "Tuition Fee : " << tuition_fee << endl;
    cout << "House Rent : " << house_rent << endl;
    cout << "EB Bill : " << *eb_bill << endl;
    cout << "Savings : " << saving;
}

void Budget::setField(string field, double value)
{
    if (field == "base")
    {
        setBase(value);
        updateSavings();
    }
    else if (field == "units_used")
    {
        setUnits(value);
        updateSavings();
    }
```

```
    else if (field == "tut_fee")
    {
        tuition_fee = value;
        updateSavings();
    }
    else if (field == "res_rent")
    {
        house_rent = value;
        updateSavings();
    }
}

void Budget::setAll(double base, double tut_fee, double res_rent, double units_used)
{
    tuition_fee = tut_fee;
    house_rent = res_rent;

    scaledVals(base, units_used);
}

int main(int argc, char const *argv[])
{
    EB_amount bill(269);
    Salary sal(24000);
    Budget budget(24000, 4000, 10000, 200);

    bill.displayBill();
    cout << endl;
    sal.displaySalary();
    cout << endl << endl;
    budget.displayBudget();

    return 0;
}
```

Output:

```
EB Bill : 1076
Salary : 51360

Income : 51360
Tuition Fee : 4000
House Rent : 10000
EB Bill : 600
Savings : 36760
```

Problem 4

Question:

> *Create a class called Employee with protected data members emp_id, name and designation. It contains the member functions to get details of employee and display them. Derive two classes Permanent and Contract from Employee class. Contract has data members num_hrs and wages_per_hr. Permanent has basic, DA, TA and HRA. Get the necessary details using member functions and display the employee details with their salary according to the given data*

Code:

```
#include <iostream>

using namespace std;

class Employee;
class Permanent;
class Contract;

class Employee
{
protected:
    string emp_id;
    string name;
    string designation;

public:
    Employee();
    Employee(string eno, string peru, string desig);

    void setFields(string eno, string peru, string desig);
    string getField(string field) const;
    void dispDetails() const;
};

class Permanent : public Employee
{
private:
    double basic;
    double DA;
    double HRA;
    double total;

public:
    Permanent();
    Permanent(double base);

    void assignBaseScaled();
    void displaySalary() const;
    void setBase(double base);
};

class Contract : public Employee
{
```

```
private:
    double num_hrs;
    double wages_per_hr;
    double pay;

public:
    Contract();
    Contract(double worked_hrs, double wages);

    void calcPay();
    void displayPay() const;
    void setPayScalers(double worked_hrs, double wages);
};

Employee::Employee()
{
    emp_id = "";
    name = "";
    designation = "";
}

Employee::Employee(string eno, string peru, string desig)
{
    emp_id = eno;
    name = peru;
    designation = desig;
}

void Employee::setFields(string eno, string peru, string desig)
{
    emp_id = eno;
    name = peru;
    designation = desig;
}

string Employee::getField(string field) const
{
    if (field == "emp_id")
    {
        return emp_id;
    }
    else if (field == "name")
    {
        return name;
    }
    else if (field == "designation")
    {
        return designation;
    }

    return "";
}
```

```
void Employee::dispDetails() const
{
    cout << "Employee ID : " << emp_id << endl;
    cout << "Name : " << name << endl;
    cout << "Designation : " << designation;
}

Permanent::Permanent()
{
    basic = 0;
    DA = 0;
    HRA = 0;
    total = 0;
}

Permanent::Permanent(double base)
{
    basic = base;

    assignBaseScaled();
}

void Permanent::assignBaseScaled()
{
    DA = 1.04 * basic;
    HRA = 0.1 * basic;
    total = basic + DA + HRA;
}

void Permanent::displaySalary() const
{
    cout << "Salary : " << total;
}

void Permanent::setBase(double base)
{
    basic = base;

    assignBaseScaled();
}

Contract::Contract()
{
    num_hrs = 0;
    wages_per_hr = 0;
    pay = 0;
}

Contract::Contract(double worked_hrs, double wages)
{
    num_hrs = worked_hrs;
```

```
    wages_per_hr = wages;
    pay = 0;

    calcPay();
}

void Contract::calcPay()
{
    pay = num_hrs * wages_per_hr;
}

void Contract::displayPay() const
{
    cout << "Salary : " << pay;
}

void Contract::setPayScalers(double worked_hrs, double wages)
{
    num_hrs = worked_hrs;
    wages_per_hr = wages;
    pay = 0;

    calcPay();
}

int main(int argc, char const *argv[])
{
    Permanent perm_emp(24000);
    Contract cont_emp(24, 3250);

    perm_emp.setFields("No#1", "Dude#1", "Developer");
    cont_emp.setFields("No#2", "Dude#2", "WH-Hacker");

    perm_emp.dispDetails();
    cout << endl;
    perm_emp.displaySalary();
    cout << endl
         << endl;

    cont_emp.dispDetails();
    cout << endl;
    cont_emp.displayPay();
    cout << endl;

    return 0;
}
```

Output:

```
Employee ID : No#1
Name : Dude#1
Designation : Developer
```

```
Salary : 51360

Employee ID : No#2
Name : Dude#2
Designation : WH-Hacker
Salary : 78000
```
