Да се дефинира класа ***Call*** во која ќе се чуваат информации за: телефонски број кон која е извршено јавувањето (стринг од 9 знаци), времетраење во секунди (цел број), датум на повик (објект од класа ***Date*** која опишува датум).

За класата да се дефинираат потребните конструктори, ***set()*** и ***get()*** методи, оператор за печатење **<<**, како и метод ***void read(ifstream& in)***, за читање на информациите за повикот од датотека што се предава како аргумент на методот (преку ***ifstream*** објектот). **(10 п.)**

Да се дефинира класа ***Log*** во која ќе се чуваат информации за: листа на повици (динамички алоцирана низа од објекти од класата ***Call***, како и цел број ***n*** кој ја означува големината на листата).

За класата ***Log*** да се имплементираат:

- Потребните конструктори и деструктор **(5 п.)**
- Метод ***void read(ifstream& in)***, за читање на информациите за класата од датотека што се предава како аргумент на методот (преку ***ifstream*** објектот). Овој метод задолжително треба да го повика методот ***read*** од класата ***Call***, за читање на информациите за листата од повици. **(5 п.)**
- Оператор за печатење **<< (5 п.)**
- Метод ***Log daily(Date d)*** - кој како аргумент прима датум, а враќа нов објект од класата ***Log*** кој во листата од повици ги содржи само оние направени на датум ***d***. **(10 п.)**

Дополнете ја *main* функцијата со следните барања:

- Од датотеката **prva.txt** прочитајте ги информациите за повиците. Секој податок се наоѓа во засебен ред (прво се зададени информациите за бројот на повици, па на крајот и информациите за секој повик што е направен). Информациите се вчитуваат со повик на методот ***read*** за објектот ***log***.
- Во датотеката ***vtora.txt*** се печатат сите информации за повиците.
- Во датотеката ***treta.txt*** се печатат информациите само за оние повици кои се добиваат како резултат од методот ***daily***. **(15 п.)**

Define a class ***Call***. The class should store information about: dialed telephone number (string of length 9), duration in seconds (integer), call date (object from class ***Date***).

For the class, define the necessary constructors, ***set()*** and ***get()*** methods, the output operator **<<**, and a method ***void read(ifstream& in)***, for reading the call information from a file passed as an argument to the method (through the ***ifstream*** object). **(10 p.)**

Define a class ***Log***. The class should store information about: list of calls (dynamically allocated array of objects of class ***Call***, and an integer ***n*** indicating the size of the list).

For the class ***Log*** implement:

- Required constructors and destructor **(5 p.)**
- Method ***void read(ifstream& in)*** for reading the class information from a file passed as an argument to the method (through the ***ifstream*** object). This method must call the ***read*** method of the class ***Call*** to read the information about the list of films. **(5 p.)**
- Output operator **<< (5 p.)**
- Method ***Log daily(Date d)*** - which takes a date as an argument, and returns a new object of class ***Log*** that contains only those calls in the list whose call date is ***d***. **(10 p.)**

Enhance the main function with the following requirements:

- Read call information from the file ***prva.txt***. Each piece of data is on a separate line (first the information about the number of calls are given and finally the information about each call. The information is read by calling the ***read*** method for the ***log*** object.

- Print all information about the calls to the file ***vtora.txt.***
- Print information about only those calls that are result from the method daily to the file ***treta.txt***. **(15 p.)**

For example:

Input	Result
10 08 11 2015 070214011 56 08 11 2018 072230807 82 01 08 2019 070214011 110 08 04 2015 071230230 152 20 04 2019 070214011 54 10 11 2018 075443026 110 19 09 2016 076215555 45 24 08 2015 070880599 87 11 06 2018 075777888 98 27 07 2019 070214011 125 ---- 8 11 2015	All the data for the log: Dialed number: 070214011 56 sec, 8/11/2015 Dialed number: 072230807 82 sec, 8/11/2018 Dialed number: 070214011 110 sec, 1/8/2019 Dialed number: 071230230 152 sec, 8/4/2015 Dialed number: 070214011 54 sec, 20/4/2019 Dialed number: 075443026 110 sec, 10/11/2018 Dialed number: 076215555 45 sec, 19/9/2016 Dialed number: 070880599 87 sec, 24/8/2015 Dialed number: 075777888 98 sec, 11/6/2018 Dialed number: 070214011 125 sec, 27/7/2019 Calls on date 8/11/2015 Dialed number: 070214011 56 sec, 8/11/2015

```
#include <iostream>
#include <cstring>
#include <fstream>
#include <vector>

using namespace std;

class Date{
private:
    int d, m, y;
public:
    Date(){}
    Date(int dd, int mm, int yy){
        d = dd;
        m = mm;
        y = mm;
    }
    ~Date(){}
    bool compare(Date &date){
        return y == date.y && m == date.m && d == date.d;
```

```
    }
    friend ostream &operator<<(ostream &out, Date & f){
        out<< f.d << "/" << f.m << "/" << f.y;
        return out;
    }
    friend istream &operator>>(istream &in, Date & f){
        in >> f.d;
        in >> f.m;
        in >> f.y;
        return in;
    }
    void read(ifstream& in){
        in >> d >> m >> y;
        in.ignore();
    }
};


class Call{
private:
    string dialedNumber;
    int duration;
    Date date;
public:
    Call(){}
    Call(const string & num, int dur, Date d){
        dialedNumber = num;
        duration = dur;
        date = d;
    }
    ~Call(){}
    void read(ifstream& in)
    {
        date.read(in);
        getline(in, dialedNumber);
        in >> duration;
        in.ignore();
    }

    friend ostream &operator<<(ostream &out, Call & f){
        out<<"Dialed number: "<<f.dialedNumber<<"
"<<f.duration<<" sec, "<< f.date << endl;
        return out;
    }
    Date getDate(){
        return date;
    }
};

class Log{
private:
    Call *calls;
    int n;
public:
    Log(){
        calls = nullptr;
        n = 0;
    }
    Log(Call *c, int num){
        n = num;
        calls = new Call[n];
        for(int i = 0; i < n; i++){
            calls[i] = c[i];
```

```
        }
    }
    Log(const Log &l){
        n = l.n;
        calls = new Call[n];
        for(int i = 0; i < n; i++){
            calls[i] = l.calls[i];
        }
    }
    ~Log(){
        delete [] calls;
    }
    void read(ifstream& in){
        in>>n;
        in.ignore();
        calls = new Call[n];
        for(int i = 0; i < n; i++)
        {
            calls[i].read(in);
        }
    }
    friend ostream &operator<<(ostream &out, Log & f){
        for(int i = 0; i < f.n; i++)
            out<< f.calls[i];
        return out;
    }
    Log daily(Date d){
        int c = 0;
        Call *tmp = new Call[n];
        for(int i = 0; i < n; i++){
            if(calls[i].getDate().compare(d)){
                tmp[c++] = calls[i];
            }
        }
        Log log(tmp, c);
        return log;
    }
};



void wtf() {
    ofstream fout("prva.txt");
    string line;
    while (getline(std::cin, line)) {
        if (line == "----"){
            break;
        }
        fout << line << endl;
    }
}

void rff(string path) {
    ifstream fin(path);
    string line;
    while (getline(fin, line)) {
        cout << line << endl;
    }
}


int main() {
```

```
    wtf();


    //TODO your code here

    ifstream fin("prva.txt");
    ofstream fout("vtora.txt");
    ofstream fout2("treta.txt");

    Log log;

    log.read(fin);

    fin.close();

    fout<<log;
    fout.close();

//DO NOT MODIFY THE CODE BETWEEN THIS AND THE NEXT COMMENT
    Date d;
    cin >> d;
//DO NOT MODIFY THE CODE BETWEEN THIS AND THE PREVIOUS COMMENT

//TODO Save the results in the files vtora.txt and treta.txt
after this line

    Log log1 = log.daily(d);
    fout2 << log1;
    fout2.close();


//DO NOT MODIFY THE CODE BELLOW

    cout << "All the data for the log:" << endl;
    rff("vtora.txt");
    cout << "Calls on date " << d << endl;
    rff("treta.txt");


    return 0;
}
```

Question text

Да се дефинира апстрактна класа **Machine** која чува информации за:

- модел (стринг)
- потрошувачка на енергија (цел број)
- основна цена (децимален број)
- сериски број (стринг)

Во рамките на оваа класа да се дефинираат два чисто виртуелни метода:

- **float calculateCost()** - за пресметка на цената
- **void showDetails()** - за печатење на информациите за една машина **(5 поени)**

Серискиот број на секоја машина е стринг со должина 12, кој мора да ја има таа должина и не смее да содржи празни места. Доколку серискиот број на една машина не ги исполнува овие услови, да се фрли исклучок од класата **InvalidSerialNumberException**. Со исклучокот треба да се, при секој обид за креирање објект. При справувањето со исклучокот да се испечати соодветна порака. **(15 поени)**.

Од класата Machine да се изведат две класи: WashingMachine и DryingMachine, кои справите во методата createMachines претставуваат машина за перење и машина за сушење, соодветно. За секоја машина за перење, дополнително се чува капацитетот на товар (цел број, во кг), а основната цена е 500$. За секоја машина за сушење, дополнително се чува дали има сензор (bool), а основната цена е 600$. **(5 поени)**

Во изведените класи соодветно да се препокријат виртуелните методи. Форматот за печатење да се види од тест-примерите, а пресметката на цената се прави на следниот начин:

- За WashingMachine - доколку капацитетот на товар е поголем од 8kg, цената се зголемува за 12%, а за секои дополнителни 100 вати потрошувачка на енергија, цената се зголемува за 20$ (пресметката се однесува на претходно зголемената цена).
- За DryingMachine - доколку машината има сензор, цената се зголемува за 15%, а за секои дополнителни 100 вати потрошувачка на енергија, цената се зголемува за 25$ (пресметката се однесува на претходно зголемената цена). **(10 поени)**

Да се дефинира глобална метода void calculatePercentageOfCost(Machine **machines, int n), која како аргументи прима низа од покажувачи кон машини и нејзината големина. Функцијата треба да го испечати процентот на цените на машините за перење и машините за сушење во вкупната цена на сите машини. (**15 поени)**

Define an abstract class **Machine** that stores information about:

- model (string)
- power consumption (integer)
- base price (decimal)
- serial number (string)

In this class, define two pure virtual methods:

- **float calculateCost()** - for calculating the cost
- **void showDetails()** - for printing the information about a machine (5 points)

The serial number of each machine is a string of length 12, which must have that length and must not contain any spaces. If the serial number of a machine does not meet these conditions, an exception of class **InvalidSerialNumberException** should be thrown. Handle this exception in the createMachines method, each time an object is created. Print an appropriate message when handling the exception. **(15 points)**

Derive two classes from Machine: **WashingMachine** and **DryingMachine**. For each washing machine, additionally store the load capacity (integer), and the base price is $500. For each drying machine, additionally, store whether it has a sensor (bool), and the base price is $600. **(5 points)**

In the derived classes, the virtual methods should be appropriately overridden. The print format should be consistent with the provided test examples, and the cost calculation is as follows:

- For **WashingMachine** - if the load capacity is greater than 8kg, the cost increases by 12%, and for every additional 100 watts of power

consumption, the cost increases by $20 (the calculation applies to the previously increased cost).
- For **DryingMachine** - if the machine has a sensor, the cost increases by 15%, and for every additional 100 watts of power consumption, the cost increases by $25 (the calculation applies to the previously increased cost). **(10 points)**

Define a global method **void calculatePercentageOfCost(Machine **machines, int n)**, which takes an array of pointers to machines and its size as arguments. The function should print the percentage of costs of washing machines and drying machines in the total cost of all machines. **(15 points)**

For example:

Input	Result
1 4 1 LG 1200 WM123456789A 10 2 Bosch 1500 DM987654321B 1 1 Samsung 1000 WM234567890C 7 2 Siemens 1800 DM345678901D 0	TESTING ABSTRACT CLASS AND CHILD CLASSES Washing Machine LG cost: 800$ Drying Machine Bosch cost: 1065$ Washing Machine Samsung cost: 700$ Drying Machine Siemens cost: 1050$ ABSTRACT CLASS AND CHILD CLASSES OK
2 2 1 LG 1200 WM123456 10 2 Bosch 1500 DM9876543212 1	TEST EXCEPTION HANDLING The serial number WM123456 is invalid machine serial number format. Drying Machine Bosch cost: 1065$ EXCEPTION HANDLING OK
3 4 1 LG 1200 WM123456789A 10 2 Bosch 1500 DM987654321B 1 1 Samsung 1000 WM234567890C 7 2 Siemens 1800 DM345678901D 0	INTEGRATION TEST AND TESTING GLOBAL FUNCTION Washing Machine LG cost: 800$ Drying Machine Bosch cost: 1065$ Washing Machine Samsung cost: 700$ Drying Machine Siemens cost: 1050$ The total cost of washing machines is 1500$, which is 41.4938% of the total cost of machines. The total cost of drying machines is 2115$, which is 58.5062% of the total cost of machines. INTEGRATION TEST AND TESTING GL

```cpp
#include <iostream>
#include <cstring>

using namespace std;


class InvalidSerialNumberFormatException {
private:
    string serialNumber;
public:
    InvalidSerialNumberFormatException(string &serialNumber) :
serialNumber(serialNumber) {}

    void printMessage() {
        cout << "The serial number " << serialNumber
             << " is invalid machine serial number format." <<
endl;
    }
};
```

```
class Machine {
protected:
    string model;
    int powerConsumption;
    float price;
    string serialNumber;

public:
    Machine(string model, int powerConsumption, string
serialNumber) {
        if (serialNumber.length() != 12 || serialNumber.find('
') != string::npos) {
            throw
InvalidSerialNumberFormatException(serialNumber);
        }
        this->serialNumber = serialNumber;
        this->model = model;
        this->powerConsumption = powerConsumption;
    }

    virtual float calculateCost() = 0;

    virtual void showDetails() = 0;
};

class WashingMachine : public Machine {
private:
    int capacity;

public:
    WashingMachine(string model, int powerConsumption, string
serialNumber, int capacity) : Machine(model,

powerConsumption,

serialNumber),

capacity(capacity) {}

    float calculateCost() {
        price = (capacity > 8) ? 500 * 1.12 : 500;
        price += (powerConsumption / 100) * 20;
        return price;
    }

    void showDetails() {
        cout << "Washing Machine " << model << " cost: " <<
calculateCost() << "$" << endl;
    }

};

class DryingMachine : public Machine {
private:
    bool sensor;
public:
    DryingMachine(string model, int powerConsumption, string
serialNumber, bool sensor) : Machine(model,

powerConsumption,

serialNumber),
```

```
sensor(sensor) {}

    float calculateCost() {
        price = sensor ? 600 * 1.15 : 600;
        price += (powerConsumption / 100) * 25;
        return price;
    }

    void showDetails() {
        cout << "Drying Machine " << model << " cost: " <<
calculateCost() << "$" << endl;
    }
};

void calculatePercentageOfCost(Machine **machines, int n) {
    float totalCost = 0.0, washingMachinesCost = 0.0,
dryingMachinesCost = 0.0;
    for (int i = 0; i < n; i++) {
        totalCost += machines[i]->calculateCost();
        (dynamic_cast<WashingMachine *> (machines[i])) !=
nullptr ? washingMachinesCost += machines[i]->calculateCost()

: dryingMachinesCost += machines[i]->calculateCost();
    }

    cout << "The total cost of washing machines is " <<
washingMachinesCost << "$, which is "
         << ((washingMachinesCost / totalCost) * 100)
         << "% of the total cost of machines." << endl;
    cout << "The total cost of drying machines is " <<
dryingMachinesCost << "$, which is "
         << ((dryingMachinesCost / totalCost) * 100)
         << "% of the total cost of machines." << endl;
}


Machine **createMachines(int &n) {
    Machine **machines = new Machine *[n];
    string model, serialNumber;
    int t, powerConsumption, value;
    for (int i = 0; i < n; i++) {
        cin >> t;
        cin >> model >> powerConsumption >> serialNumber >>
value;

        try {
            (t == 1) ? machines[i] = new WashingMachine(model,
powerConsumption, serialNumber, value)
                     : machines[i] = new DryingMachine(model,
powerConsumption, serialNumber, value);
        } catch (InvalidSerialNumberFormatException e) {
            e.printMessage();
            n--;
            i--;
        }
    }
    return machines;
}


void cleanUp(Machine **machines, int n) {
    for (int i = 0; i < n; i++) {
```

```
        delete machines[i];
    }
    delete[] machines;
}


int main() {
    int testCase, n;

    cin >> testCase;
    switch (testCase) {
        case 1: {
            cout << "TESTING ABSTRACT CLASS AND CHILD CLASSES"
<< endl;
            cin >> n;
            Machine **machines = createMachines(n);
            for (int i = 0; i < n; i++) {
                machines[i]->showDetails();
            }
            cleanUp(machines, n);
            cout << "ABSTRACT CLASS AND CHILD CLASSES OK" <<
endl;
            break;
        }
        case 2: {
            cout << "TEST EXCEPTION HANDLING" << endl;
            cin >> n;
            Machine **machines = createMachines(n);
            for (int i = 0; i < n; i++) {
                machines[i]->showDetails();
            }
            cleanUp(machines, n);
            cout << "EXCEPTION HANDLING OK" << endl;
            break;
        }
        default: {
            cout << "INTEGRATION TEST AND TESTING GLOBAL
FUNCTION" << endl;
            cin >> n;
            Machine **machines = createMachines(n);
            for (int i = 0; i < n; i++) {
                machines[i]->showDetails();
            }
            calculatePercentageOfCost(machines, n);
            cleanUp(machines, n);
            cout << "INTEGRATION TEST AND TESTING GLOBAL
FUNCTION OK" << endl;
            break;
        }
    }
}
```