Question

Answered step-by-step

Part 1 (0%)

This workshop consists of two modules:

- w1 (partially supplied)
- Food Order

w1 Module (partially supplied)

Study the code supplied and make sure you understand it.

Finish the implementation of the main function, by completing the parts marked with TODO:

- write the prototype of the main function to receive a set of standard command line arguments
- echo the set of arguments to standard output in the following format:

```
1: first argument
2: second argument
3: third argument
4: ...
```

Do not modify this module in any other place!

FoodOrder Module

The FoodOrder module defines the current tax rate and daily special discount, as two global variables named g_taxrate and g_dailydiscount that stores double values. The value of the tax rate and daily discount may change from day to day. These two variables will be accessed when a new day of orders starts from the main function.

This module also defines a class named FoodOrder in the namespace sdds that stores some information about an order of food:

a C-style null-terminated string of up to 10 characters *including the null byte terminator* representing the name of the customer ordering the food.

a C-style null-terminated string of up to 25 characters *including the null byte terminator* representing the description of the food.

the price of the food represented by a double value.

a boolean flag indicating whether or not the order is a daily special

Public Members

Default constructor

read(): a modifier that receives an istream reference.

- If the istream is in a good state, this function will attempt to read from the istream and store the information read into current object. The data read in will be in the following format:

```
<Customer Name>,<Order Description>,<Price>,<Daily Special Status>[newline]
```

- The daily special status can be of two values: 'Y' for it is a special and 'N' for it is not.
- If the istream object is not in a good state then this function will do nothing.

display(): a query that displays to the screen the content of an Food Order instance in the following format:

```
COUNTER. Name          |Order Description        |Price w/Tax |Special Price
```

If no customer name has been stored in the current object, this query should print:

```
COUNTER. No Order
```

where

- COUNTER is a *left aligned* field of size 2, that holds the number of times that this function has been called (use a local-to-function variable that remains in memory for the lifetime of the program) **Do not use global/member variables to store the counter!**
- Name is a *left aligned* field of size 10 that represents the name of the customer that made the order.
- Order Description is a *left aligned* field of size 25 that represents the description of the food ordered.
- Price w/Tax is a *left aligned* field of size 12 with 2 decimal point precision that represents the price of the food ordered with tax calculated based on the current tax rate.

- Special Price is a *right aligned* field of size 13 that represents the taxed price of the food ordered with the daily discount applied on top if the order was a daily special. If the order isn't a daily special then nothing is printed here.

Add any other **private** members that your design requires (without changing the specs above)!

Sample Output

The input files day1.txt and day2.txt are already provided; the main module contains a description of the structure for these files.
When the program is started with the command:

```
ws day1.txt day2.txt
```

the output should look like the one from the sample_output.txt file.

Test Your Code

To test and demonstrate execution of your program use the same data as shown in the output example above.
Upload your source code to your matrix account. Compile and run your code using the latest version of the g++ compiler (available at /usr/local/gcc/10.2.0/bin/g++) and make sure that everything works properly.
Then, run the following command from your account (replace profname.proflastname with your professor's Seneca userid):

```
~profname.proflastname/submit 345_w1_p1
```

and follow the instructions.

This part represents a milestone in completing the workshop and is not marked!

Part 2 (100%)

For this part of the workshop, upgrade your FoodOrders class to accept an food description in the form of a C-style null-terminated string of any length. **Make sure your program doesn't have memory leaks.**
In order to facilitate the changes needed for this part, consider the use of these std::string related functions:

- std::getline
- std::string::c_str
- std::string::length

Sample Output

When the program is started with the command:

```
ws day1.txt missing.txt day2.txt
```

the output should look like the one from the sample_output.txt file.

Reflection

Study your final solution, reread the related parts of the course notes, and make sure that you have understood the concepts covered by this workshop. **This should take no less than 30 minutes of your time and the result is suggested to be at least 150 words in length.**
Create a **text** file named reflect.txt that contains your ***detailed*** description of the topics that you have learned in completing this workshop and mention any issues that caused you difficulty and how you solved them. Include in your explanation—**but do not limit it to**—the following points:

- the difference between internal and external linkage citing examples from your code
- what are static variables and how are they useful in your solution.
- the changes that you made in upgrading your FoodOrder class in part 2.

Computer Science | Engineering & Technology | C++ Programming | CPD OOP345

Answer & Explanation

Solved by verified expert

Rated Helpful

ProteinDatabase.cpp

```
#include <fstream>
#include <iostream>

#include"ProteinDatabase.h"

using namespace std;

namespace sdds {
    ProteinDatabase::ProteinDatabase() {

    }

    ProteinDatabase::ProteinDatabase(const ProteinDatabase& obj) {
        if (obj.m_strings != nullptr) {
            this->m_strings = new string[obj.m_noOfStrings];

            unsigned int i;
            for (i = 0; i < obj.m_noOfStrings; i++) {
                this->m_strings[i] = obj.m_strings[i];
            }

            this->m_noOfStrings = obj.m_noOfStrings;
        }
    }

    ProteinDatabase& ProteinDatabase::operator=(const ProteinDatabase& obj) {
        if (this != &obj) {
            this->m_strings = new string[obj.m_noOfStrings];

            unsigned int i;
            for (i = 0; i < obj.m_noOfStrings; i++) {
                this->m_strings[i] = obj.m_strings[i];
            }

            this->m_noOfStrings = obj.m_noOfStrings;
        }

        return (*this);
    }

    ProteinDatabase::~ProteinDatabase() {
        delete[] m_strings;
    }

    ProteinDatabase::ProteinDatabase(ProteinDatabase&& src) noexcept {
        *this = std::move(src);
    }

    ProteinDatabase& ProteinDatabase::operator=(ProteinDatabase&& src) noexcept {
        if (this != &src) {
            delete this->m_strings;
            this->m_strings = src.m_strings;
            src.m_strings = nullptr;

            this->m_noOfStrings = src.m_noOfStrings;
            src.m_noOfStrings = 0;

        }

        return (*this);
    }

    ProteinDatabase::ProteinDatabase(const char* fileName) {
        string line;

        std::ifstream f(fileName);

        // i. reads the file to count the number of protein sequence present in the file.
        unsigned int proteinCount = 0;
        if (f.is_open()) {
            while (getline(f, line))
```

```
            {
                if (line[0] == '>')
                    proteinCount++;
            }

            m_noOfStrings = proteinCount;
            f.close();
        }

        // ii. allocates memory for that number of protein sequences in the array
        m_strings = new string[m_noOfStrings + 1];

        // iii. re-reads the file and loads the protein sequences (i.e., string of characters not sequence names) into
the array.
        proteinCount = 0;
        f.open(fileName);
        if (f.is_open()) {
            while (getline(f, line) && proteinCount < m_noOfStrings)
            {
                if (line[0] != '>') {
                    m_strings[proteinCount] += line;
                }

                if (line[0] == '>' && !m_strings[proteinCount].empty()) {
                    proteinCount++;
                }
            }

            f.close();
        }
    }

    std::size_t ProteinDatabase::size() {
        return m_noOfStrings;
    }

    std::string ProteinDatabase::operator[](size_t i) {
        if (i > m_noOfStrings - 1 || i < 0 || m_noOfStrings == 0) {
            // Invalid index
            return "";
        }

        return m_strings[i];
    }
}
```

ProteinDatabase.h

```
#ifndef SDDS_PROTEINDATABASE_H
#define SDDS_PROTEINDATABASE_H

#include<string>

namespace sdds {

    class ProteinDatabase {
        std::string *m_strings{};
        unsigned int m_noOfStrings = 0;

    public:
        ProteinDatabase();

        // RULE OF FIVE : Copy Constructor, User Defined Copy Assignment Operator, Destructor, Move Constructor, Move
Operator
        ProteinDatabase(const ProteinDatabase& obj);
        ProteinDatabase& operator=(const ProteinDatabase& obj);
        ~ProteinDatabase();
        ProteinDatabase(ProteinDatabase&& obj) noexcept;
        ProteinDatabase& operator=(ProteinDatabase&& src) noexcept;



        ProteinDatabase(const char* fileName);

        std::size_t size();

        std::string operator[](size_t);
    };
}

#endif
```

Step-by-step explanation

Explanation in the comment part of the code.

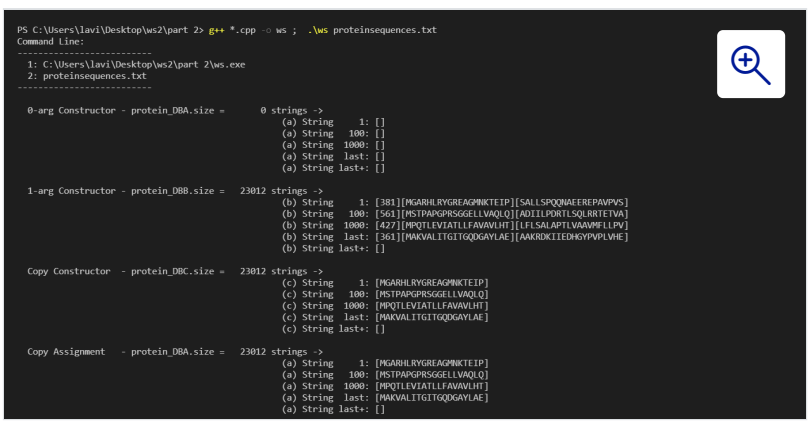


Image transcription text

PS C: \Users\lavi\Des... Show more

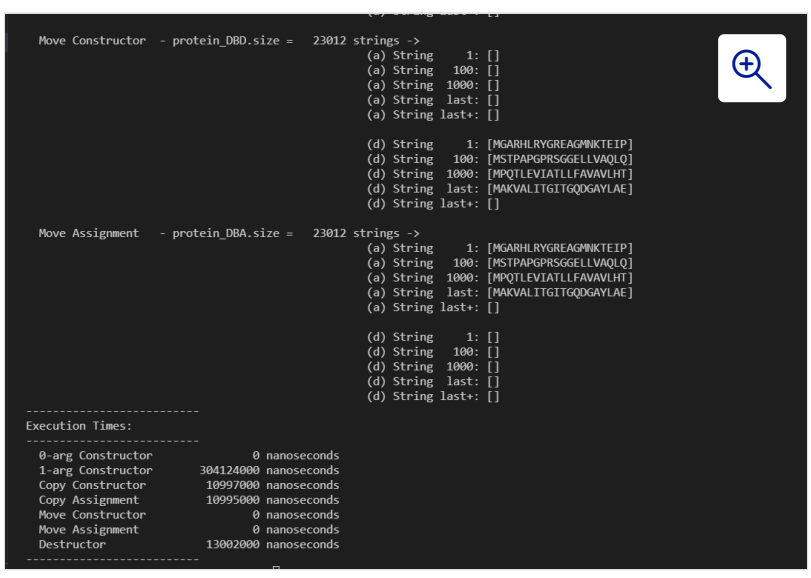


Image transcription text

Move Constructor - protein_DBD. size = 23012 strings -> (a) Stri... Show more

Student review 100% (1 rating)

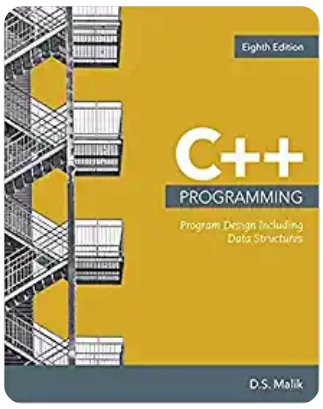
Thorough explanation Easy to follow Clear formatting

Is this answer helpful? **Helpful** **Unhelpful**

Report this answer

Add to library

We have related textbook solutions for you!



This textbook contains questions and solutions related to the question you are viewing.

Chapter 9 / Exercise 4

C++ Programming: Program Design Including Data Structures

Malik

Expert Verified

View Solutions

Other related textbook solutions

Browse all textbooks solutions

Related Course Resources

CPD 244
Seneca College
3988 Documents
985 Question & Answers

OOP 345
Seneca College
1984 Documents
272 Question & Answers

Seneca C
846 D
105 Q

Company Get Course Hero Careers