СУ „Св. Климент Охридски“, ФМИ

Специалност „Софтуерно Инженерство“

Обектно-ориентирано програмиране, 2020-2021 г.

Задача за домашно № 2

Срок: 18.04.2021 г. 23:59

Важна информация

Инструкции

1. *Позволено е използването на всички библиотеки от STL*
2. *Не променяйте предоставяните публични интерфейси (методи и полета) на класовете, тъй като тези методи ще се използват в автоматични тестове и ако имат променена сигнатура тестовете няма да компилират и ще получите 0 точки. От вас се очаква да имплементирате дадените методи.*
3. *За да компилира кодът ви трябва всички методи да имат имплементация, дори да връщат грешен отговор.*
4. *Позволено е да добавяте други методи/класове, за да реализирате задачата. Тях няма да ги тестваме.*
5. *Не е позволено използването на наследяване в домашното или на външни библиотеки (които не са част от STL)*

Оценяване на домашното

- Част от точките за това домашно ще бъдат давани след покриването на автоматични тестове - за **коректно реализирана функционалност**
- За да получите тези точки, предадените от вас решения трябва да отговарят на следните критерии
    - Да съдържат указаните методи и имена на класове (ще ви бъде даден шаблон, върху който да работите) - позволено е да добавяте нови методи и класове, но **не е позволено** да променяте даденото от нас.
    - Предавайте единствено файлове съдържащи код - архиви съдържащи `.sln` файлове или каквито и да е други файлове, които не са `.cpp` или `.hpp` ще получават 0 точки на автоматичните тестове.
    - **Не предавайте** архиви от тип `.rar` - **ще се приемат архиви от тип `.zip`.** При получен архив от тип `.rar` (или друг тип, които не

може да бъде разархивиран от системата за тестване), отново получавате 0 точки на автоматичните тестове.
    - Именувайте архива си по следния начин - `SI_R_HW2_<курс>_<група>_<факултетен номер>`. Архиви, които не спазват тази конвенция ще получат 0 точки на автоматичните тестове. (Пример: `SI_R_HW2_1_1_12345.zip`)
    - Не променяйте имената на файлове, които получавате
    - След разархивиране на архива, трябва да се получат 3 папки, с имена ‘1’, ‘2’ и ‘3’
    - Може да тествате архивите си тук: [линк]
    - Спазвайте следната структура на архива:

```
SI_R_HW2_<курс>_<група>_<факултетен номер>.zip
├── 1
│   ├── Error.hpp
│   ├── Error.cpp
│   ├── Optional.hpp
│   └── Result.hpp
├── 2
│   ├── ElectionResultsDatabase.cpp
│   ├── ElectionResultsDatabase.hpp
│   ├── SectionVotes.cpp
│   └── SectionVotes.hpp
└── 3
    ├── Vector4D.cpp
    └── Vector4D.hpp
```

- Ако решението на някоя задача ви не се компилира, получавате 0 точки на автоматичните тестове за съответната задача
- Спазвайте практиките за обектно-ориентирано програмиране, коментирани на упражнения и лекции.

Задача 1 (4 точки - 3 точки от автоматични тестове)

Линк към шаблона: [тук]

Условие

Съществуват няколко начина за обработване на грешки в различните езици за програмиране. Тук ще разгледаме един от тях.

Error

Даден е клас `Error`, който пази в себе си съобщение за грешка.

Optional

Даден е клас `Optional<T>`, който може да съдържа даден елемент, но може и да е празен.

Проверката дали даден `Optional` обект е празен, се извършва от метода `is_none()`

Можем да вземем стойността на `Optional` обект (дори и да е празен), чрез метода `get_value()` - ако обекта е празен, пак се очаква да върнем обект празен от тип `T` .

Един `Optional<T>` е празен, когато не му подадем аргумент в конструктора.

Result

Даден е клас `Result<T>`, който може да съдържа даден резултат, или пък грешка (от тип `Error`).

Предефинирайте оператора за сравнение (==), така че да приема `Error` или `T` и да връща дали дадения `Result` обект е грешка или не.

Напишете методи, за връщане на резултата и грешката, като използвате `Optional` (все пак, може и да нямаме резултат, или пък да нямаме грешка) .

Допълнение: Няма да се инстанцират Result<Error> и Result<std::string>
Пример за оператор== при Result:
Result<int>(3) == int() -> true
Result<int>(5) == Error() -> false
Result<int>(“Error Message”) == Error() -> true

Задача 2: (4 точки - 3 точки от автоматични тестове)

Линк към шаблона: тук

Условие

За да елиминира ръчното броене на бюлетини и неработещите машини за гласуване, държавата е поръчала на вас да направите софтуер, който да брои и сумира автоматично гласовете на всяка партия от всички секции в страната и чужбина.

(За целите на задачата, приемаме, че в страната има само три партии с имената PARTY1, PARTY2, PARTY3.)

Новите машини за гласуване ще съхраняват бройките получени гласове в текстов файл със следния формат:

```
{SECTION1_PARTY1_VOTES} {SECTION1_PARTY2_VOTES} {SECTION1_PARTY3_VOTES}
{SECTION2_PARTY1_VOTES} {SECTION2_PARTY2_VOTES} {SECTION2_PARTY3_VOTES}
{SECTION3_PARTY1_VOTES} {SECTION3_PARTY2_VOTES} {SECTION3_PARTY3_VOTES}
{SECTION4_PARTY1_VOTES} {SECTION4_PARTY2_VOTES} {SECTION4_PARTY3_VOTES}
...
```

т.е. един **ред** пази гласовете, получени в една изборна **секция**, като на всеки ред с **един интервал** са отделени цели числа, представляващи бройката гласове за съответната партия в тази секция (по 3 цели числа на ред, понеже имаме 3 партии).

Пример за тесктови данни във файл с такъв формат:

```
5 1 2
6 10 12
20 24 8
14 15 18
5 10 0
```

Тези данни ни казват, че в секция първа има постъпили 5 гласа за PARTY1, 1 глас за PARTY2 и 2 гласа за PARTY3; в секция втора има по 6, 10 и 12 гласа за съответните партии и т.н. Общо секциите, от които има информация в този файл са 5 на брой.

От вас се изисква да създадете два класа - `SectionVotes` и `ElectionResultsDatabase`.

SectionVotes

Класът трябва да пази информация за подадените гласове за всяка от партиите в **една** изборна секция. Трябва да съдържа:

- Конструктор с 3 параметъра - гласовете в тази секция за PARTY1, PARTY2 и PARTY3 съответно
- `int votesForParty(Party) const` - връща колко гласове е събрала дадената партия в тази секция

За да може тази информация да бъде лесно прочетена или записана във файл **с формат като горепосочения**, трябва да бъдат предефинирани операторите `<<` и `>>` за работа със съответните потоци.

ElectionResultsDatabase

Класът трябва да борави с файлове с формат като горепосочения и да пази информация за всички постъпили гласове във всички секции. Има следните методи:

- `void addResultsFromFile(const char* filename)`: Прочита информация, съдържаща се във файл с име `filename` и с формат като горепосочения.
    - **Не трябва да се трие вече съществуващата информация в класа, а само да се добавя към нея**
    - Съдържанието на файла не трябва да бъде променяно по никакъв начин
    - Очаквайте файловете да бъдат само с коректно форматирани данни
- `int votesForParty(Party) const`: Връща колко гласове общо е събрала дадената партия
- `Party winningParty() const`: Връща партията с най-много гласове от изборите.
    - Ако PartyX и PartyY (X < Y) имат еднакъв брой гласове, то в този случай се очаква да върне PartyX
- `int numberOfSections() const`: Връща от колко изборни секции има информация за гласуването

За лесната работа с този тип файлове, чийто формат е посочен по-горе в условието, трябва и да се предифинират операторите `<<` и `>>` за работа със съответните потоци:

- Операторът за четене от поток не трябва да изтрива съществуващите данни в класа, а само добавя прочетените такива от целия поток.
- Операторът за писане в поток трябва да копира в потока абсолютно всички данни, пазещи се в класа, във формат като горепосочения.

Важно: Декларации на операторите, които трябва да се предефинират, не са ви дадени в хедър файла - трябва вие да си ги напишете правилно. При пропуск на някой от тях решението няма да се компилира и ще получи 0т. на автоматичните тестове.

Задача 3: (2 точки - 1.9 точки от автоматични тестове)

Линк към шаблона: тук

Условие

Даден е клас `Vector4D`, представляващ наредена четворка с реални числа. Добавете декларации и дефиниции на необходимите оператори, така че да бъдат възможни следните операции по следните правила:

- Достъп

```
Vector4D v = Vector4D(a, b, c, d);
v[0] == a;
v[1] == b;
v[2] == c;
v[3] == d;

// Мутирането също трябва да е възможно:
v[0] = x; // вече v е наредената четворка (x, b, c, d)
```

- Събиране
  - `Vector4D(a, b, c, d) + Vector4D(i, j, k, l) == Vector4D(a+i, b+j, c+k, d+l)`
  - Аналогично за `+=`
- Изваждане
  - `Vector4D(a, b, c, d) - Vector4D(i, j, k, l) == Vector4D(a-i, b-j, c-k, d-l)`
  - Аналогично за `-=`
- Умножение
  - Поелементно умножение: `Vector4D(a, b, c, d) * Vector4D(i, j, k, l) == Vector4D(a*i, b*j, c*k, d*l)`
  - Умножение със скалар (double) **отдясно**: `Vector4D(a, b, c, d) * x == Vector4D(x*a, x*b, x*c, x*d)`
  - Аналогично за `*=`
- Деление
  - `Vector4D(a, b, c, d) / Vector4D(i, j, k, l) == Vector4D(a/i, b/j, c/k, d/l)`
  - Аналогично за `/=`
- Сравнение
  - `==` и `!=`: За да са равни две наредени четворки, трябва да съвпадат поелементно
  - `<`, `>`, `<=`, `>=`: Сравняват елементите лексикографски
  - Освен това нека и `!v == true` тогава и само тогава, когато `v == Vector4D(0, 0, 0, 0)`
- Отрицание
  - `-Vector4D(1, 2, 3, 4) == Vector4D(-1, -2, -3, -4)`

Важно: Ако не дефинирате някой от изброените оператори получавате 0т. от макс. 1.9т. на тестовете за тази задача.