Предварителна задача по Обектно-ориентирано програмиране - 2024

Съгласно условието на задачата е нужно да се създаде абстрактен и шаблонен клас, източник на данни, *data_source*, чийто данни са от произволен тип. Всеки елемент от тези данни се извлича от източника и се преминава към следващия.

Обектите, представители на наследници от този клас позволяват:

> Извличане на един елемент, ако има. Това се случва чрез метода *next()*, който е функция с чисто виртуално поведение, и всеки от наследниците го реализира по свой собствен начин;

> Извличане накуп на определен брой поредни елементи, ако има толкова. Б.О.О. нека са *n*. Това е реализирано чрез метода *next_n()*. Когато наличните елементи са по-малко от *n*, се извличат колкото са налични. Ако $n < 1$, то се хвърля грешка - няма как да се изведат отрицателен или нулев брой елементи. Стойността, която се връща, е от тип *s_vector<T>;*

> Проверка дали има следващ елемент. Това става чрез функцията *end().* Ако върне истина, то гарантирано следващото извличане на елемент успява, тъй като методът *next()* съдържа в себе си проверка дали е достигнат краят на източника. Също е с чисто виртуално поведение и всеки от наследниците я реализира по свой собствен начин;

> Метод *reset(),* който възстановява началното състояние на източника. Работата на тази функция се улеснява - "преминава към следващ елемент", а не "премахва" елементите - по идея на доц. П. Армянов (1). Но възстановяването не е винаги възможно. По-долу е обяснено кога. Връща булева стойност - дали операцията е била успешна или не. Има чисто виртуално поведение и всеки от наследниците, както *next()* и *end()* по-горе, я реализира по свой собствен начин;

> *operator(),* който извлича и връща като резултат един елемент - чрез *next();*

> *operator>>,* който извлича елемент в десния си аргумент. Позволява слепване *(src >> x >> y).* Реализира се отново чрез *next();*

> *operator bool(),* който връща истина, ако обектът може да генерира още елементи и лъжа в противен случай. Това става чрез *end().*

Съдържа и метод:

> *clone()* - при полиморфизъм, необходим при създаване на копия на обекти, без да се знае точният им тип по време на компилация. Чисто виртуално поведение.

Реализирани са следните конкретни наследници:

*default_data_source : public data_source<T>*

> Връща безкрайно много подразбиращо конструирани обекти от типа на данните си;

> *next()* връща подразбиращо се конструиран обект от тип T;

> *end()* винаги връща лъжа. Това е по дизайн, тъй като от условието се изисква да се връщат "безкрайно много" обекти от тип T т.е. никога няма край;

> *reset()* винаги връща истина. Това също е по дизайн след съгласуване с доц. П. Армянов (2);

*file_data_source : public data_source<T>*

> Създава се чрез символен низ - име на текстов файл. Елементите се четат последователно от този файл;

> Съдържа поток и символен низ, от тип *std::string*, като член-данни;

> *next()* връща последователно елемент от даден файл;

> *end()* проверява дали е достигнат края на файла;

> *reset()* възстановява началното състояние на източника т.с.т.к. файлът е в коректно състояние (3);

*array_data_source : public data_source<T>*

> Създава се чрез масив от елементи и връща последователно тези елементи;

> Има масив *s_vector<T>* и индекс, който се грижи за последователното преминаване през елементите на масива, като член-данни;

> *next(),* ако не е изчерпан източника, връща последователно елемент от източника;

> *end()* проверява дали е достигнат края на масива;

> *reset()* връща индекса на нулева позиция;

> *operator+* и *operator+=* добавят елемент в края на масива;

> Префиксен и постфиксен *operator--* връщат източника един елемент назад.

*alternate_data_source : public data_source<T>*

> Създава се чрез масив от източници на данни;

> Подобно на *array_data_source*, съдържа масив *s_vector<data_source<T>*>* и индекс, който се грижи за последователното преминаване през елементите на масива, като член-данни;

> *next()* извлича последователно елементи от тях - първо от първия, после от втория, после от третия и т.н. и след извличане на елемент от последния подаден източник отново преминава към елемент от първия и т.н.

> *end()* проверява дали всички източници-елементи са изчерпани;

> *reset()* възстановя всеки източник-елемент поотделно и връща индекса на нулева позиция. Ако при някоя от елементите възникне грешка, връща лъжа - съгласувано с доц. П. Армянов;

*generator_data_source : public data_source<T>*

> Създава се от конструктор с аргумент генератор с поведение на функция;

> Има функтор като член-данна;

> *next()* използва функцията за генериране на елементите;

> *end()* винаги връща лъжа по дизайн, тъй като това е безкраен генератор и няма "край";

> *reset()* винаги връща лъжа, отново по дизайн и идея на доц. П. Армянов (4), защото за рестартиране на последователността от някакви елементи, трябва да се знае какъв е генератора;

*s_vector<T>*

> Реализация на динамичен масив, подобен на *std::vector*;

> Този клас е създаден да управлява колекция от елементи от произволен тип данни T, като има способността автоматично да променя размера си, когато е необходимо повече място.

Демонстрацията за използването на класовете се случва в два етапа:

> Създава се източник *generator_data_source<std::string>* чрез функцията *generate_strings()* - всеки низ е с дължина точно 10 символа и на екрана се извеждат 25 низа извлечени от този източник;

> Създават се три източника:

>> *generator_data_source<int>* чрез функцията *consecutive_prime_numbers();*
>> *generator_data_source<int>* чрез функцията *consecutive_random_real_numbers(),* която сама по себе си използва *std::rand(),* съгласувано с доц. П. Армянов (5);
>> *generator_data_source<int>* чрез *consecutive_fibonacci_numbers()*, след това се създава *array_data_source<int>*, който се запълва с първите 25 числа на Фибоначи, използвайки *next_n();*

> Създава се *alternate_data_source<int>* от горепосочените източника;

> Записват се 1000 числа от този източник в бинарен файл с подадено от потребителя име. След това се прехвърлят от този файл в текстов файл, създава се файлов източник от него и се извеждат на екрана всички елементи през този файлов източник. Всичко това е съгласувано с доц. П. Армянов.

Използвани библиотеки: *<iostream>, <fstream>, <stdexcept>* и *<cstring>.*

# Въпроси, свързани с лятната домашна по ООП-Редовен

**Stoyan Ivanov** <sustoyanivanov2@gmail.com> 31 юли 2024 г. в 11:32
До: Петър Армянов <armianov@gmail.com>

Здравейте,

Имам няколко въпроса, свързани с лятната домашна за редовния курс по ООП.

1) Във втората подзадача за демонстрация на работата на класовете:

Идеята ми е да имам отделна функция, примерно да я наречем test(), която да съдържа generator_data_source( foo). Съответно, функцията foo() да има в себе си три generator_data_source - за просто число, за случайно цяло число и за число на Фибоначи. При първото достъпване на следващ елемент
на generator_data_source(foo) в test() трябва да се "даде" просто число, при второто - случайно цяло, и при третото - число на Фибоначи. И така да се редуват. Това струва ли Ви се правдоподобно? Защото условието "алтернативно връща поредно" така го разбирам.

```
static int counter = 0;

int foo()
{
   generator_data_source<int> gds_p(prime);
   generator_data_source<int> gds_r(real);
   generator_data_source<int> gds_f(fib);

   ++counter;

    if(counter % 3 == 1) return gds_p.next();
    else if(counter % 3 == 2) return gds_r.next();
    else return gds_f.next();
}

void test()
{
   generator_data_source<int> gds(foo);
   gds.next(); // просто
   gds.next(); // случайно цяло
   gds.next(); // число на Фибоначи
}
```

2) Също свързано с втората задача, може ли за генериране на случайни цели числа просто да използвам std::rand()? Първоначално си мислех да използвам <ctime> библиотеката, но тя не е позволена по условие.

3) Reset кога не е успешен? Двете неща, за които се сетих в последния месец, е reset да не работи при default_data_source, защото той така или иначе генерира безкрайно много еднакви по подразбиране елементи, т.е. какъвто и елемент да искаме, той ще е същият и reset няма смисъл да "reset"-ва каквото и да е. Но тогава защо просто да не връща true - не е станала някаква промяна. Също reset не би трябвало да работи при generator_data_source, защото той извиква пореден елемент от някаква генерираща функция. Ако е така, тогава reset трябва да връща false или мога да хвърля грешка, която казва, че тази функция е unsupported?

4) За втората подзадача за демонстрация на работата на класовете, функцията за генериране на низове трябва да връща просто някакви низове с дължина 10 случайни символа, примерно uasiopdsfb, нали? Не е нужно да са някакви смислени думи?

Благодаря предварително за отговора!

**Петър Армянов** <armianov@gmail.com> 31 юли 2024 г. в 16:01
До: Stoyan Ivanov <sustoyanivanov2@gmail.com>

Здравей,
За 1) по-скоро си представям да си направите обект от тип AlternateDataSource - точно това е идеята на този клас. Иначе може да го сглобиш от един ArrayDataSource с числата на фибоначи, един GeneratorDataSource за простите числа и вече за случайните можеш да го направиш или пак с GeneratorDataSource или с DefaultDataSource, ако си направиш подходящ тип (клас).
2) rand е супер за целта.
3) Тук две неща - нямаш гаранция какво прави DefaultDataSource - например ако по подразбиране обектът се създава със случайна стойност, то няма да са еднакви. Все пак може да приемеш, че тук reset винаги работи. От друга страна за Generator (често) нямаш възможност да за reset. Например как ще рестартираш последователността от прости числа, ако не знаеш какъв е генератора? Тук е ОК да върнеш false (аз бих предпочел този интерфейс, вместо хвърляне на изключения). Същото важи и за файла - ако не позволява seek (представи си cin) . От там и сглобените класове (AlternateDataSource) зависят от това какво имат вътре.
4) Точно така - просто случайни низове.
Пиши, ако нещо не съм успял да отговоря или има още неясноти.
Поздрави,
Петър Армянов

**Stoyan Ivanov** <sustoyanivanov2@gmail.com> 31 юли 2024 г. в 16:56
До: Петър Армянов <armianov@gmail.com>

Здравейте,

По първия въпрос: Щом идеята е да се използва alternate_data_source и трябва да е съставен от три data_source-а, тогава generator_data_source има ли "край"? Винаги има следващ елемент. Тогава, когато трябва да запиша 1000 числа в двоичен файл, както е по условие, ако винаги има следващ елемент, то просто ще се запишат първите 1000 числа от първия генератор и никога няма да достигне втория, третия и т.н, защото той е един вид безкраен. Би ли било правилно в alternate_data_source да имам метод, да кажем move_to_next_element(), който да сменя на кой елемент в alternate_data_source съм? Така, примерно, ще запиша 487 прости числа, ще преместя към следващия генератор, ще запиша 488 случайни числа, ще преместя и накрая ще запиша 25 числа на Фибоначи?

И докато пиша този имейл се сетих още един въпрос:

5) Отново по втората подзадача за демонстрация на работата на класовете: Тези 1000 числа трябва да се запишат в двоичен файл и после чрез file_data_source да се прочетат от текстов файл. Това означава ли, че първо трябва да отворя бинарния файл, да запиша информацията временно и след това тази информация да я прехвърля в текстов файл? Или е нещо съвсем друго, което в момента не мога да се сетя?

Благодаря,
Стоян Иванов

**Петър Армянов** <armianov@gmail.com> 31 юли 2024 г. в 23:16
До: Stoyan Ivanov <sustoyanivanov2@gmail.com>

alternate_data_source по условие трябва да дава елементи алтернативно от източниците си - един от първия, един от втория, един от третия, после пак от първи, от втори... докато някой (примерно втори) свърши. Тогава дава един от първи, един от трети, един от първи...
След като имаш двоичния файл с числата как те ще стигнат до текстовия не е особено важно в тази задача. Може да ги прочетеш всичките в масив и да ги изсипеш в текстовия файл или да ги пренасяш едно по едно... както ти е удобно. Фокус е да покажеш, че можеш да работиш със съответните типове файлове.
Поздрави,
Петър
Петър Армянов

**Петя Личева** 07/19/2024 5:31 PM
Здравейте!
Имам 2 въпроса относно лятната задача по ООП и те са следните:

1. Като извлека един елемент, това означава ли, че той се маха изобщо от контейнера, в който съхраняваме въпросните данни? Например, ако имаме 1 -> 2 -> 3 -> nullptr и искаме да извлечем 2, това означава ли, е трябва да ъпдейтнем данните в контейнера на 1 -> 3 -> nullptr?
2. В задачата оператор >> трябва да се пренапише така, че от даден източник да можем да извличаме определени елементи (например, ако имаме източника source със следната структура 1 -> 2 -> 3 -> 4 -> 5 -> 6 -> nullptr и приложим оператор >>: source >> 1 -> 3 -> 5 трябва ли да получим 2 -> 4 -> 6 -> nullptr) или се има нещо друго предвид?

Благодаря Ви предварително за съдействието! (edited)

Петър Армянов 07/19/2024 6:47 PM
1) Не е казано какво правите вътрешно. Важно е да се извличат един по един.
Аз лично не бих ги махал, най-малкото за да мога лесно да връщам началното състояние. (1)
2) трябва да се извличат с оператора както с функцията. Просто за удобство вместо да пиша
src.get(x);
src.get(y);
src.get(z);
да напиша
src >> x >> y >> z;
Или както прецениш да е метода за извличане