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;

> Mетод *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 Т;

> *end()* винаги връща лъжа. Това е по дизайн, тъй като от условието се изисква да се връщат "безкрайно много" обекти от тип Т 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. флагът за файла *fail* е свален (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>.*



