Практичне заняття

Об’єктно-орієнтований підхід

Мета: засвоєння навичок створення та роботи із складними структурами даних з використанням об’єктно-орієнтованого підходу на мові С++

Об’єктно-орієнтований підхід дає можливість суттєво спростити процес створення складних програм, які оперують із складними структурами даних. Основна ідея полягає у тому, що на початку створюється загальна концепція структур даних та функціоналу у вигляді базових абстрактних класів. На основі цих класів створюються породжені, які перевизначають операції базових класів. Такий підхід дає можливість максимально уникнути дублювання структур даних так коду. Потужним загальновживаним застосуванням об’єктного підходу є фабрики і шаблони проектування. Шаблон проектування – це ієрархія класів, які породжені від одного базового, мають визначений функціонал і мають віртуальну функцію створення нової копіє об’єкту будь-якого типу. Фабрика – це контейнер, який містить асоційований масив об’єктів всіх можливих типів і може динамічно створювати об’єкти будь-якого з підтримуваних типів у процесі роботи.

Приклад програми, у якій використовується фабрика для динамічного створення структур даних типу int та string за результатами читання стандартного вводу.

#include <iostream>

#include <map>

#include <string>

#include <list>

#include <iterator>

#include <algorithm>

using namespace std;

//abstract base class

struct data\_elem {

virtual data\_elem\* create() = 0;

virtual ostream& print(ostream&) = 0;

virtual istream& read(istream&) = 0;

};

//creator class

template <typename elem\_name, typename data\_type>

struct data\_creator:public data\_elem{

data\_type data;

data\_elem\* create() {return new elem\_name;}

ostream& print (ostream& o){

return o<<data;

}

istream& read(istream& i){

return i>>data;

}

};

//data classes

struct string\_data: public data\_creator<string\_data,string>{};

struct int\_data: public data\_creator<int\_data,int>{};

struct fabric: public map<string, data\_elem\* > {

fabric (){

(\*this)["string"]= new string\_data;

(\*this)["int"]= new int\_data;

}

data\_elem\* create(string& name){

data\_elem \*& d = (\*this)[name];

if(!d){

erase(find(name)->first);

return NULL;

} else {

return d->create();

}

}

virtual ~fabric(){

for (pair<string,data\_elem\*> p: (\*this)){

if(p.second) delete p.second;

}

}

};

struct my\_data\_base: public list<data\_elem\*>{

virtual ~my\_data\_base(){

for (data\_elem\* d: (\*this)){

if(d) delete d;

}

}

};

fabric f;

istream& operator >>(istream& i,data\_elem \*& d){

string name;

if( i>>name && (d=f.create(name)) && d->read(i)) {

return i;

} else {

cerr << "error at: "<<name<<endl;

d=NULL;

return i;

}

}

ostream& operator <<(ostream& o, data\_elem\* d){

if(d) return d->print(o);

else return o;

}

int main(int argc, char\*\* argv){

my\_data\_base b;

copy\_if(istream\_iterator<data\_elem\*>(cin),istream\_iterator<data\_elem\*>(), back\_inserter(b), [](data\_elem\* d){return d!=NULL;});

copy(begin(b), end(b), ostream\_iterator<data\_elem\*> (cout,"\n"));

}

У програмі визначено абстрактний базовий клас data\_elem, який задає загальний функціонал всіх динамічних структур даних, зокрема чисті (нульові) віртуальні методи create для створення структур даних типу data\_elem, та методи print та read для їх вводу та вводу на стандартні потоки.

Клас data\_creator є шаблонним, який породжений від класу data\_elem, тобто він має всі властивості (зокрема і тип) класу data\_elem і додає новий функціонал. Параметрами шаблону є тип даних elem\_name (якого типу буде новий контейнер), а також тип даних data\_type, який у цей контейнер вкладається. Функції створення, вводу та виводу відповідно виконують вказані раніше дії для об’єктів типів шаблонних параметрів (ввід- виввід поля data) і створення нового об’єкту типу elem\_name., який є також має і тип data\_elem ,оскільки породжений від нього.

Далі визначаються типи даних породжені від шаблону. У нашому випадку string\_data і int\_data. Для цього застосовується прийом рекурсивного шаблону, тобто клас породжується від шаблону, який має параметром цей же породжений тип. Другий параметр – тип даних у контейнері. Рекурсивний шаблон дозволяє створити віртуальні конструктори. Зверніть увагу на невелику кількість коду для створення нових типів даних.

Далі визначено клас фабрику fabric , задача якого динамічно створювати об’єкти різних типів. Цей клас наслідує асоційований масив ключами якого є назви об’єктів, а даними – вказівники на об’єкти відповідного класу. У конструкторі створюються примірники класів string\_data з ключем "string" та int\_data з ключем "int". Функція цього класу create створює об’єкти відповідних класів за їх іменами. Для цього викликається віртуальна функція create відповідного об’єкта у асоційованому масиві. Оскільки функція create віртуальна, то вона при виклику «знає» тип об’єкта для якого виконується, незважаючи на те, що всі вказівники на об’єкти мають тип data\_elem\* . Це найефективніше застосування поліморфізму. Також у класі виизначено деструктор для видалення динамічних структур.

Далі визначено оператори вводу і виводу (>>,<<) структур data\_elem за вказівником на них. Зверніть увагу, що визначено лише один оператор вводу і один оператор виводу на «всі випадки життя», тобто для всіх структур даних, які похідні від data\_elem .Оператор вводу зчитує ім’я типу яке треба створити, звертається до фабрики для створення об’єкту потрібного типу і передає цьому об’єкту потік вводу. Далі об’єкт зчитує себе сам. У разі неуспішного створення чи читання нічого не відбувається потік починає інтерпретуватись далі. Вивід здійснюється ще простіше.

Також визначено контейнер my\_data\_base, який містить всі вдало введені дані і є похідним від стандартного контейнера.

Тут використано об’єктний підхід для мінімізації кількості коду (один оператор на всіх), зменшення кількості помилок (більшість помилок відпрацює компілятор, інші локалізовані у невеликій кількості коду) та спрощення програмування складних операцій. Такий підхід є вдалим поєднанням низькорівневого і високорівневого підходів для комбінації ефективності і простоти. Зверність увагу, як ,використовуючи наслідування, можна змінювати поведінку об’єктів мінімально переписуючи існуючий код.

У функції main заповнюється контейнер об’єктів різних типів із стандартного вводу за допомогою стандартного алгоритму copy\_if . Вхідним ітератором є istream\_iterator , який читає об’єкти типу data\_elem\* із стандартного вводу, вихідним ітератором є ітератор вставки у кінець контейнера , а умовою є лямбда-вираз перевірки поточного вказівника на нуль. Таким чином вставляються лише успішно прочитані об’єкти. Далі список об’єктів виводиться на стандартний вивід.

Приклад компіляції програми

[saa@plus7 tmp]$ g++ -std=c++11 file.cpp

Приклад роботи програми:

(echo "int 3"; echo "string hhhhh"; echo "10"; echo "int 10")| ./a.out

error at: 10

error at:

3

hhhhh

10

# Завдання

1. Скомпілюйте і запустіть програму з прикладу
2. Розберіться, що ця програма робить і як реалізована. Зокрема, як реалізований і навіщо потрібен абстрактний базовий клас data\_elem ; навіщо потрібен і як реалізовано клас віртуального конструктора data\_creator ; як на основі віртуального конструктора описуються типи різних даних, які можна створювати в процесі роботи програми (string\_data, int\_data); яку структуру даних має фабрика і навіщо вона потрібна; як фабрика створює копії об'єктів у реальному часі; як вводяться дані за допомогою оператора вводу і як виводяться за допомогою оператора виводу
3. Модифікуйте програму з практичного завдання “3 Алгоритми і структури даних” для запису даних цієї ж лабораторної роботи у файл у форматі

ім”я значення

наприклад, 1-й рядок має бути представлений у вигляді

Rank 1

NCT\_Number NCT03689842

Title Feasibility Study of Uterine Transplantation From Living Donors in Terms of Efficacy and Safety in Patients With Mayer-Rokitansky-Küster-Hauser Syndrome (MRKH)

….

за ним ідуть дані другого рядка і так далі. Кожен елемент даних з нового рядка . Пусті елементи даних і ті, що не важливі у індивідуальному завданні, не повинні записуватись у файл

Колонки даних мають імена

"Rank","NCT Number","Title","Acronym","Status","Study Results","Conditions","Interventions","Outcome Measures","Sponsor/Collaborators","Gender","Age","Phases","Enrollment","Funded Bys","Study Type","Study Designs","Other IDs","Start Date","Primary Completion Date","Completion Date","First Posted","Results First Posted","Last Update Posted","Locations","Study Documents","URL"

1. На основі прикладу розробіть клас-фабрику, яка створює об”єкти для колонок, що важливі у індивідуальному завданні. Максимально використовуйте наслідування і віртуальні функції для мінімізації кількості вашої роботи. За допомогою фабрики реалізуйте читання даних із файла у форматі ім”я значення
2. Виведіть гістограми відповідно до індивідуального завдання (номер відповідає номеру студента у групі). Продемонструйте працездатність системи. Дайте відповіді на контрольні питання

## Індивідуальні завдання

1. Створіть хеш-таблички даних за статтю і віком і їх комбінаціями. Підрахуйте кількість пацієнтів різних статей і і віку. Виведіть отриману гістограму розподілів за віком
2. Створіть хеш-таблички даних за назвою обстеження і віком і їх комбінаціями. Підрахуйте кількість пацієнтів з різними назвами і віком. Виведіть отриману гістограму розподілів за назвами від віку
3. Створіть хеш-таблички даних за локаціями і віком і їх комбінаціями. Підрахуйте кількість пацієнтів з різними титулами і віком. Виведіть отриману гістограму розподілів за локаціями від віку
4. Створіть хеш-таблички даних за типом обстеження і віком і їх комбінаціями. Підрахуйте кількість пацієнтів з різними типами обстеження і віком. Виведіть отриману гістограму розподілів за типами обстежень від віку
5. Створіть хеш-таблички даних за спонсорами-колабораторами і статтю і їх комбінаціями. Підрахуйте кількість пацієнтів з різними колабораторами і статтю. Виведіть отриману гістограму розподілів за колабораторами і статтю
6. Створіть хеш-таблички даних за тривалістю обстеження і статтю і їх комбінаціями. Підрахуйте кількість пацієнтів з різними тривалостями обстежень і статтю. Виведіть гістограму розподілів за тривалостями обстеження і статтю
7. Створіть хеш-таблички даних за результатми і віком і їх комбінаціями. Підрахуйте кількість пацієнтів з різними результатми обстежень і віком. Виведіть отриману гістограму розподілів за результатми обстеження і віком
8. Створіть хеш-таблички даних за interventions і віком і їх комбінаціями. Підрахуйте кількість пацієнтів з різними interventions і выком. Виведіть отриману гістограму розподілів за результатами обстеження і віком
9. Створіть хеш-таблички даних за outcome measures і віком і їх комбінаціями. Підрахуйте кількість пацієнтів з різними interventions і выком. Виведіть отриману гістограму розподілів за outcome measures і віком
10. Створіть хеш-таблички даних за Study designs і віком і їх комбінаціями. Підрахуйте кількість пацієнтів з різними interventions і выком. Виведіть отриману гістограму розподілів за study design і віком

## Контрольні питання

1. Які переваги і недоліки об’єктно-орієнтованого підходу?
2. Які переваги і недоліки має вирішення поставленої задачі з використанням фабрики класів порівняно з тим, як це робилось лиже з використанням алгоритмів і структур даних STL на практичному занятті 2?
3. Що треба зробити у вашій програмі, щоб додати ще одну колонку даних, яка містить посилання на зображення?

Література

1. Курси з обєктно-орієнтованого програмування <https://www.udemy.com/course/intermediate-cpp/?utm_source=adwords&utm_medium=udemyads&utm_campaign=LongTail_la.EN_cc.ROW&utm_content=deal4584&utm_term=_._ag_77879424134_._ad_535397245863_._kw__._de_c_._dm__._pl__._ti_dsa-1007766171312_._li_9061016_._pd__._&matchtype=&gclid=CjwKCAjw4P6oBhBsEiwAKYVkq4E7ttp21bcksiv7VPTdZaKvp2GACDFL7vcvQoB8OamOnShikRxBIRoCZk0QAvD_BwE>
2. Фабрика класів <https://simpledevcode.wordpress.com/2015/12/07/class-factory-pattern-c/>