Лекція 15. Огляд можливостей Standard Templates Library.

1. Поняття про узагальнене програмування.

2. Призначення та структура бібліотеки STL.

3. STL. Використання контейнерів. Ітератори.

4. Алгоритми і функтори.

1. Поняття про узагальнене програмування

Узагальнене програмування (англ. generic programming) – парадигма програмування, яка полягає в описі даних та алгоритмів, які можна застосовувати до різних типів даних, не змінюючи сам опис. Узагальнене програмування, у тому чи іншому вигляді підтримується різними мовами програмування. Можливості узагальненого програмування вперше з'явилися у вигляді дженериків (узагальнених функцій) у 1970-х роках у мовах CLU та Ада, потім у вигляді параметричного поліморфізму у ML та його нащадках, а потім у багатьох об'єктно-орієнтованих мовах, таких як C++, Python, Java, Object Pascal, D, Eiffel, мовах платформи .NET та інших.

Узагальнене програмування є методологією програмування, що грунтується на поділі структур даних і алгоритмів через використання абстрактних описів вимог. Абстрактні описи вимог є розширенням поняття абстрактного типу даних. Замість опису окремого типу в узагальненому програмуванні застосовується опис сімейства типів, що мають загальний інтерфейс та семантичну поведінку (англ. semantic behavior). Набір вимог, що описує інтерфейс та семантичну поведінку, називається концепцією (англ. concept). Таким чином, написаний в узагальненому стилі алгоритм може застосовуватися для будь-яких типів, що задовольняють його концепції.

Говорять, що тип моделює концепцію (є моделлю концепції), якщо він задовольняє її вимогам. Вимоги до концепцій містять таку інформацію:

Допустимі вирази (англ. valid expressions) – вирази мови програмування, які мають успішно компілюватися для типів, що моделюють концепцію.

Асоційовані типи (англ. associated types) – допоміжні типи, що мають стосуються моделюючого концепцію типу.

Інваріанти (англ. invariants) - такі характеристики типів, які повинні бути незмінними під час виконання. Зазвичай виражаються як передумови і постумови. Невиконання передумови тягне за собою непередбачуваність відповідної операції і може призвести до помилок.

Гарантії складності (англ. complexity guarantees) - максимальний час виконання допустимого виразу або максимальні вимоги до різних ресурсів під час виконання цього виразу.

У С++ ООП реалізується у вигляді віртуальних функцій та успадкування, а узагальнене програмування – з допомогою шаблонів класів і функцій. Проте суть обох методологій пов'язана з конкретними технологіями реалізації лише побічно. Говорячи формально, ООП засноване на поліморфізмі підтипів, а узагальнене програмування на параметричному поліморфізмі. В інших мовах те й інше може бути реалізоване інакше.

Виділяють такі етапи у розв'язанні задачі з методології узагальненого програмування:

- Знайти корисний та ефективний алгоритм.
- Визначити узагальнене уявлення (параметризувати алгоритм, мінімізувавши вимоги до даних, що обробляються).
- Описати набір (мінімальних) вимог, задовольняючи які ще можна отримати ефективні алгоритми.
- Створити каркас з урахуванням класифікованих вимог.

Мінімізація та створення каркасу ставлять за мету створення такої структури, при якій алгоритми не залежать від конкретних типів даних. Цей підхід застосовано у структурі бібліотеки STL.

2. Призначення та структура бібліотеки STL

Бібліотека стандартних шаблонів (***STL***) (англ. ***Standard Template Library***) – набір узгоджених узагальнених алгоритмів, контейнерів, засобів доступу до їхнього вмісту та різних допоміжних функцій у С++.

Бібліотека стандартних шаблонів до включення до стандарту С++ була сторонньою розробкою, спочатку фірми HP, а потім SGI. Стандарт мови не називає її «STL», оскільки ця бібліотека стала невід'ємною частиною мови, проте багато людей досі використовують цю назву, щоб відрізняти її від решти стандартної бібліотеки (потоки вводу-виводу (iostream), підмножина Сі і ін.).

Проект під назвою STLPort, заснований на SGI STL, здійснює постійне оновлення STL, iostream та рядкових класів. Деякі інші проекти займаються розробкою приватних застосувань стандартної бібліотеки для різних конструкторських завдань. Кожен виробник компіляторів С++ обов'язково постачає деяку реалізацію цієї бібліотеки, оскільки вона є дуже важливою частиною стандарту та широко використовується.

Архітектура STL була розроблена Олександром Степановим та Менг Лі.

У бібліотеці виділяють п'ять основних компонентів:

Контейнер (англ. container) – зберігає набір об'єктів у пам'яті.

Ітератор (англ. iterator) – забезпечує засоби доступу до вмісту контейнера.

Алгоритм (англ. algorithm) – визначає обчислювальні процедури.

Адаптер (англ. adaptor) – адаптує компоненти забезпечення різного інтерфейсу.

Функціональний об'єкт (англ. functor) – інкапсулює функції в об'єкті для використання іншими компонентами.

Поділ дозволяє зменшити кількість компонентів. Наприклад, замість написання окремої функції пошуку елемента для кожного типу контейнера забезпечується єдина версія, яка працює з кожним з них, доки дотримуються основних вимог. Standard Template Library – набір узгоджених узагальнених алгоритмів, контейнерів, засобів доступу до їхнього вмісту та різних допоміжних функцій у C++.

3. STL. Використання контейнерів. Ітератори.

Контейнери бібліотеки STL можна розділити на чотири категорії: послідовні, асоціативні, контейнери-адаптери та псевдоконтейнери. У контейнерах зберігання елементів використовується семантика передачі об'єктів за значенням. Іншими словами, при додаванні контейнер отримує копію елемента. Якщо створення копії небажане, використовують контейнер покажчиків на елементи. Присвоєння елементів реалізується з допомогою оператора присвоєння, а знищення відбувається з допомогою деструктора. В бібліотеці реалізовані такі класи контейнерів, як ***vector, list, deque, stack, map, set, bitset*** і ін. Часто до контейнерів STL долучають також клас ***string***, що використовується для зберігання та опрацювання текстових стрічок замість масиву типу ***char***. Оскільки, починаючи з стандарту C++ 11, бібліотека STL включена в основу мови, то такий поділ цілком умовний.

У бібліотеці STL для доступу до елементів в якості посередника використовується узагальнена абстракція, іменована ітератором. Кожен контейнер підтримує «свій» вид ітератора, який є «модернізованим» інтелектуальним покажчиком та «знає» як отримати доступ до елементів конкретного контейнера. Стандарт C++ визначає п'ять категорій ітераторів: вхідні, вихідні, однонаправлені, двонаправлені та ітератори довільного доступу.

Методи контейнерів та ітератори дозволяють динамічно додавати-вилучати та переглядати елементи в контейнері, що робить їх застосування вигідною альтернативою до використання звичайних, чи динамічних масивів. Розглянемо невеликий приклад:

```
#include <iostream>

#include <list>

#include <algorithm>

#include <functional>

#include <Windows.h>

using namespace std;

void main() {

    SetConsoleOutputCP(1251);
```

```
    SetConsoleCP(1251);
    //оголошуємо контейнер - список цілих чисел
    list<int> numbers;
    cout << "Введіть послідовність додатних чисел "
        << "0 - для завершення\n";
    int el;
    do {
        cin >> el;
        //додаємо черговий елемент в кінець списку
        numbers.push_back(el);
    } while (el > 0);
    numbers.pop_back();
    //забираємо останній елемент (0)
    transform(numbers.begin(), numbers.end(),
 numbers.begin(), negate<int>());
    for_each(numbers.begin(), numbers.end(),
 [=](int p){cout << p << '\t'; });
}
```

Тут використовується контейнер ***list<int>*** з назвою ***numbers***, тобто динамічний список цілих чисел. Програма пропонує користувачеві ввести довільну послідовність цілих чисел, завершивши ввід числом 0, потім міняє значення чисел в списку на протилежні і виводить їх на консоль.

4. Алгоритми і функтори

В прикладі вище для опрацювання списку використано алгоритми ***transform*** та ***for_each***. Алгоритми STL реалізовані у вигляді глобальних функцій, які працюють із використанням ітераторів. Це означає, що кожен алгоритм потрібно реалізувати лише один раз, і він буде працювати з усіма контейнерами, які надають набір ітераторів (включаючи і контейнерні класи, розроблені користувачем). Хоча це має величезний потенціал і надає можливість швидко писати складний код, алгоритми також мають і «темну сторону» – деяка комбінація алгоритмів і типів контейнерів може не працювати, чи працювати з поганою продуктивністю, або викликати нескінченні цикли, тому слід бути обережним. Бібліотека STL надає багато

алгоритмів, для їх роботи необхідно підключити заголовний файл ***algorithm***.

Функтор (або функціональний об'єкт) – це будь-який об'єкт, використання якого можливе подібно до виклику функції. У термінах С++ функції не є об'єктами, тому вони функторами не вважаються, хоча взагалі у програмуванні функції часто відносять до окремих випадків функторів.

Функціональні об'єкти використовують тоді, коли функції мають поводитися як об'єкти. У С++ функтори займають важливе місце у стандартній бібліотеці шаблонів (STL). Наприклад, в алгоритмі сортування необхідно використовувати певний критерій для порівняння, роль цього критерію бере на себе функтор. Основна перевага функціональних об'єктів-класів у порівняння з функціями і вказівниками на них – можливість зберігати внутрішній стан. Адже всередині класу можна описати багато незалежних змінних та/або об'єктів. Це дозволяє робити аналог функції, який можна налаштувати в процесі роботи.

В прикладі вище ми скористалися функтором ***negate<int>()*** для заміни кожного елемента списку numbers протилежним числом за допомогою алгоритму ***transform***. В алгоритм for_each для виводу елементів списку на друк передано лямбда-вираз ***[=](int p){cout << p << '\t'; }***. Лямбда-вирази, це реалізація функції в місці її безпосереднього виклику, їх можна використовувати замість вказівників на функції та функторів.