Створення бібліотеки функціональних метапрограм мовою C++

Виконав: Трохимчук Артем Андрійович [1]
Науковоий керівник: Бублик Володимир Васильович [2]

[1]студент 3-го року навчання,
спеціальності: 121 Інженерія програмного забезпечення

[2]доцент, кандидат фізико-математичних наук

Зміст

Зміст

Завдання роботи

- Дослідити та описати методи та засоби метапрограмування в мові шаблонів, яка є частиною ядра C++.
- Дослідити та застосувати інтерфейси стандартної бібліотеки, які використовуються під час метапрограмування в C++.
- Продемонструвати використання метапрограмування для підвищення швидкодії програми, а також читабельності коду.
- Дослідити на порівняти на практиці час та використання ресурсів програмами, одна з яких використовує метапрограмування, а інша ні.

Зміст

Концепція метапрограмування

Визначення метапрограмування

Метапрограмування — це один із видів програмування, в якому програма створює чи модфікує іншу програму, базуючись на певних метаданих.

В найпростішому вигляді метапрограмування і є описом метаданих, які будуть впливати на генерацію бінарного коду.

- Метапрограмування було наявне вже у другій мові програмування високого рівня — у LISP.
- В багатьох інтерпритованих мовах метапрограмування реалізовано за рахунок гомоіконності, властивість мови програмування маніпулювати програмою як даними.

Види метапрограмування

1. Динамічне метапрограмування
2. Статичне метапрограмування

Динамічне метапрограмування

- Дуже гнучке: немає принципових обмежень
- Відносно дороге: всі обчислення відбуваються під час виконання програми
- Потребує динамічне середовище виконання: типово динамічне метапрограмування реалізовано в інтерпритованих мовах

Динамічне метапрограмування: приклад Ruby

Один із прикладів метапрограмування в Ruby — відкриті класи, будь-який код може модифікувати клас, додавши в нього, наприклад, метод.

```ruby
class String
        def shout
                self.upcase + "!"
        end
end


puts "hello".shout # => "HELLO!"
# HELLO!
```

- Код використовує одну з особливостей Ruby — відкриті класи
- В Ruby код може під час виконання додати в клас певний метод
- Модифікувати можна навіть класи базової бібліотеки

Статичне метапрограмування

- Набагато менш гнучке
- Метапрограма виконуєтсья під час компіляції
- Не накладає умови на середовище виконання: під час виконання програми ніякої ’метапрограми’ не існує

Зміст

Статичне метапрограмування на C++

Метапрограмування в C++ реалізована за допомогою шаблонів (templates) і ґрунтується на трьох методиках:

1. Спеціалізація (повна чи часткова) шаблонів
2. Особливості інстанціювання шаблонів в C++
3. Ознаки (traits) типів в STL

Мова шаблонів

Мова шаблонів — це ”мова в мові”, частина C++, яка відповідає за узагальнене програмування. Одною із задач шаблонного програмування було витіснення макросів для написання коду, додатково, мова шаблонів принесла метапрограмування в C++.

Спеціалізація шаблонів

```
1  template<typename T> struct
       TemplateSpecialization {
2      static constexpr char MSG[]{"non
       specialized :("};
3  };
4  template<> struct TemplateSpecialization<int> {
5      static constexpr char MSG[]{"specialized :)
       "};
6  };
7  int main() {
8      std::cout << TemplateSpecialization<char>::
       MSG << '\n'; // non specialized :(
9      std::cout << TemplateSpecialization<int>::
       MSG << '\n'; // specialized :)
10 }
```

- Спеціалізація шаблону — це спосіб змінити шаблонний код, залежно від шаблонних аргументів
- Спеціалізація шаблону впливає на код, який генерує компілятор із шаблонного коду
- Спеціалізувавши клас, можна змінити значення його статичних членів

Особливості інстанціювання шаблонів

SFINAE

SFINAE (читається як ”сфінае”, від англ. ”substitution failure is not an error”, ”невдала підстановка не є помилкою”) — правило інстанціювання шаблонів в C++ згідно з яким помилка під час інстанціювання не вважається помилкою в програмі.

```
template <int n> std::string create_array(char(*)[n >= 0] = nullptr) {
            return "created array of size " + std::to_string(n);
}
template <int n> std::string create_array(char(*)[n < 0] = nullptr) {
       return "cannot create array of negative size :(";
}
int main() {
       std::cout << create_array<10>() << '\n'; // created array of size 10
       std::cout << create_array<-3>() << '\n'; // "cannot create array of
      negative size :(
}
```

Можливості STL

В C++ стандартна бібліотека шаблонів (STL) була значно розширена, зокрема, в межах STL була реалізована бібліотека метапрограмування, яка складаєтсья з трьох компонентів:

1. Компонент ознак типу
2. Відношення часу компіляції
3. Цілочисельна послідовність часу компіляції

Зміст

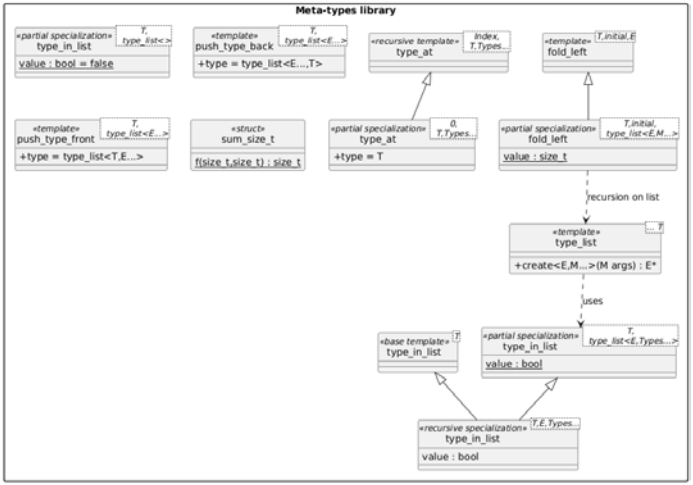
Вимоги до функціональної бібліотеки метапрограм

1. Оптимізація часу виконання
2. Підвищення типобезпеки
3. Розширення можливостей стандартної бібліотеки
4. Читабельність та орієнтованість на користувача

Компоненти біблыотеки

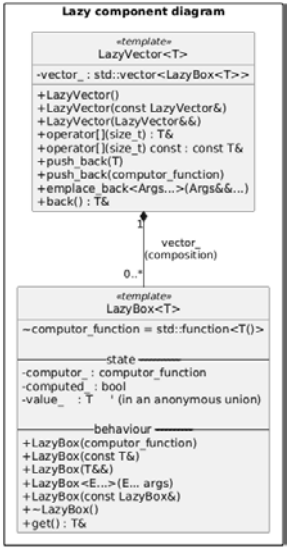
1. Компонент списку типів
2. Компонент лінивх обчислень
3. Компонент зіставлення зі зразком

Компонент списку типів



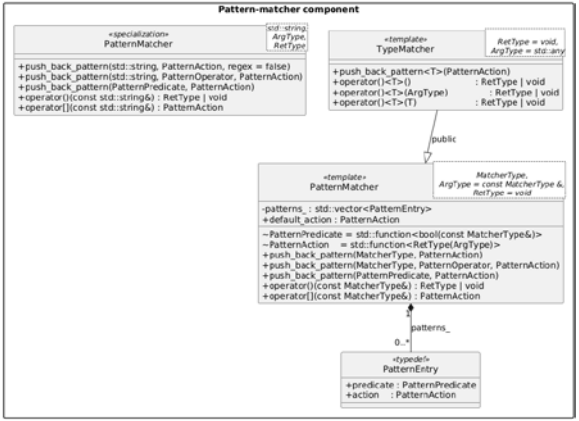
- На відміну від стандартного типу tuple, кортежу, ця структура даних не зберігає значення, а лише типи
- Список типів можна використовувати при реалізації патернів створення об’єктів
- Список типів декларативний, він явно нотує, наприклад, такі типи подій, які функція може обробити

Компонент лінивх обчислень



- Ліниві обчислення вже присутні в C++, наприклад
  (a < 100) ^& (is_fibonacci(a))
- Проте в ядрі C++ і STL їм не приділілили багато часу
- В бібліотеці надано можливість створювати функтори, які будуть обчислювати результат за потреби
- Обчислений результат буде зберігатись для подальших звернень
- Компонент реалізує два інтерфейси:
  - контейнер для одного елемента, значення якого обчислюється лінвио
  - контенйер для вектора ”лінивх” елементів

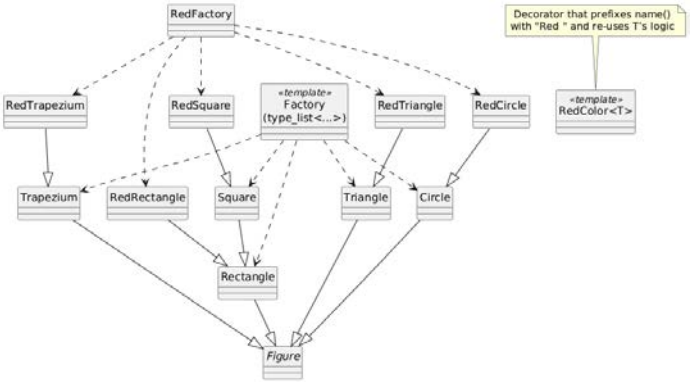
Компонент зіставлення зі зразком



- Зіставлення зі зразком характерно для мов декларативної парадигми (наприклад Haskell)
- Бібліотека реалізує три типи зіставлення зі зразком:
  - Зіставлення по типу
  - Зіставлення по значенню
  - Зіставлення для стрічок (із використанням регулярних виразів)
- Деструктурування не підтримане — в C++ бракує рефлексії

Зміст

Використання списку типів для реалізації шаблону проектування Abstract Factory



- Андрей Александреску писав ”було б добре мати factory.Create<T>();, але це не можливо через відсутність варіативних шаблонів”
- Завдяку модулю списків типів така фабрика може бути реалізована
- Функція здатна задавати власний інтерфейс, може залежати від частини фабрики

Пришвидшення програми заміною однієї стрічки коду

```
1 std::vector<User> users;
2 LazyVector<User> users; //10+ times faster
```

- Компонент лінивих обчислень довзоляє використовувати ”лінвий вектор” замість звичайного
- В прикладі читається база даних і заноситься в вектор користувачів
- Заміна звичайного вектора лінивим пришвидшила виконння коду в рази.

Використовування зіставлення зі зразком для обробки користувацького вводу

- >>> users
  - Повертає список усіх користувачів
- >>> users[:id]
  - Повертає користувача по id
- >>> users[column_name = value]
  - Повертає список користувачів, де значення в колонці column_name дорівнює value

- Для обробки користувацького вводу використовується порівняння стрічок за регулярними виразами
- Чимось нагадує ”Search Path” в USP.

Зміст

Висновки

- Виділено концепцію метапрограмування як частину шаблонного програмування в C++
- Розглянуті методи метапрограмування в C++
- Побудована функціональна бібліотека метапрограмування і приклади використання кожного її компоненту

Дякую!

Дякую за увагу, буду радий вашим запитанням та зауваженням