Рефератна частина до завдання Регулярні вирази за допомогою бібліотеки <regex> на C++11

C++ — мова програмування високого рівня з підтримкою кількох парадигм програмування: об'єктно-орієнтованої, узагальненої та процедурної. Розроблена Б'ярном Страуструпом в AT&T Bell Laboratories (Мюррей-Хілл, Нью-Джерсі) 1979 року та початково отримала назву «Сі з класами». Згодом Страуструп перейменував мову на C++ у 1983р. Базується на мові програмування C.

У 1990-х роках C++ стала однією з найуживаніших мов програмування загального призначення. Мову використовують для системного програмування, розробки програмного забезпечення, написання драйверів, потужних серверних та клієнтських програм, а також для розробки розважальних програм, наприклад, відеоігор.

Назва «Сі++» була вигадана Ріком Масситті і вперше було використана в грудні 1983 року. Раніше, на етапі розробки, нова мова називалася «Сі з класами». Ім'я, що вийшло у результаті, походить від оператора Сі «++» (збільшення значення змінної на одиницю) і поширеному способу присвоєння нових імен комп'ютерним програмам, що полягає в додаванні до імені символу «+» для позначення поліпшень. Згідно зі Страуструпом, «ця назва указує на еволюційну природу змін Сі». Виразом «C+» називали ранішню, не пов'язану з Сі++, мову програмування.

Деякі програмісти на Сі можуть відмітити, що якщо виконуються вирази x=3; y=x++; то в результаті вийде x=4 і y=3, тому що x збільшується тільки після присвоєння його y. Проте якщо другий вираз буде y=++x; то вийде x=4 і y=4. Виходячи з цього, можна зробити висновок, що логічніше було б назвати мову не Сі++, а ++Сі. Проте обидва вирази c++ і ++c збільшують c, а крім того вираз c++ поширеніший.

У серпні 2011 завершилася тривала епопея з прийняттям нового стандарту для мови Сі++. Комітет ISO зі стандартизації C++ одноголосно затвердив специфікацію C++0X як міжнародний стандарт «C++11». Стандарт C++0X планувалося випустити ще в 2008 році, але його прийняття постійно відкладалося. Більшість представлених в стандарті можливостей вже підтримуються в таких компіляторах, як GCC, IBM C++, Intel C++ і Visual C++. Підтримуючі C++11 стандартні бібліотеки реалізовані в рамках проекту Boost.

Новий стандарт розвивався понад 10 років і прийшов на зміну стандартам C++98 і C++03. Відзначається, що якщо відмінності між стандартами C++98 і C++03 були настільки незначними, що їх можна було не помітити, то стандарт C++11

містить низку кардинальних покращень, як самої мови, так і стандартної бібліотеки. За словами Б'ярна Страуструпа, творця C++, C++11 відчувається як нова мова, частини якої краще поєднуються одна з одною. У C++11 високорівневий стиль програмування став природнішим. Крім того, мова стала простішою для вивчення.

Саме 11 стандарт вже дозволяє використання бібліотеки <regex> для роботи з регулярними виразами. Тож детальніше розглянемо цю бібліотеку.

В програмуванні, регулярний вираз — це рядок, що описує або збігається з множиною рядків, відповідно до набору спеціальних синтаксичних правил. Вони використовуються в багатьох текстових редакторах та допоміжних інструментах для пошуку та зміни тексту на основі заданих шаблонів. Багато мов програмування підтримують регулярні вирази для роботи з рядками. Наприклад, Perl та Tcl мають потужний механізм для роботи, вбудований безпосередньо в їхній синтаксис. Завдяки набору утиліт (разом з редактором sed та фільтром grep), що входили до складу дистрибутивів UNIX, регулярні вирази стали відомими та поширеними.

Регулярні вирази надзвичайно корисні для вилучення інформації з тексту, наприклад коду, файлів журналів, електронних таблиць або навіть документі. Перше, що слід усвідомити при використанні регулярних виразів, це те, що по суті все є символом , і ми пишемо шаблони, щоб відповідати певній послідовності символів. У більшості шаблонів використовується звичайний ASCII, який включає літери, цифри, розділові знаки та інші символи на клавіатурі, такі як% # $ @!, Але символи Unicode також можуть використовуватися для відповідності будь-якому типу міжнародного тексту.

Регулярні вирази базуються на теорії автоматів та теорії формальних мов. Ці розділи теоретичної кібернетики займаються дослідженням моделей обчислення (автомати) та способами описання та класифікації формальних мов.

Регулярний вираз (часто називається шаблон) є послідовністю, що описує множину рядків. Ці послідовності використовують для точного описання множини без переліку всіх її елементів. Наприклад, множина, що складається із слів «грати» та «ґрати», може бути описана регулярним виразом «[гґ]рати». В більшості формалізмів, якщо існує регулярний вираз, що описує задану множину, тоді існує нескінченна кількість варіантів, які описують цю множину.

Бібліотеки <regex> не є надто функціональна, але дозволяє виконувати основні потреби. Для початку використання необхідно зробити «#include <regex>»

Для роботи з регулярними виразами в C ++ використовується клас regex, якому в якості аргументу передається рядок, що містить регулярними виразами.

Більшість посилань на C ++ говорять так, ніби C ++ 11 реалізує регулярні вирази, як визначено стандартами ECMA-262v3 та POSIX. Але насправді реалізація C ++ дуже вільно базується на цих стандартах. Синтаксис досить близький. Єдині суттєві відмінності полягають у тому, що std :: regex підтримує класи POSIX навіть у режимі ECMAScript, і що є дещо своєрідним щодо того, які символи повинні бути екрановані (наприклад, фігурні дужки та закриваючі квадратні дужки), а які не слід екранувати (як букви) .

Але існують важливі відмінності у фактичній поведінці цього синтаксису. Каретка і долар завжди збігаються на впроваджених розриви рядків в std::regex, в той час як в JavaScript і POSIX це варіант. Зворотні посилання на групи, що не беруть участь, не збігаються, як у більшості варіантів регулярних виразів, тоді як у JavaScript вони знаходять відповідність нульової довжини. У JavaScript \ d та \ w призначені лише для ASCII, тоді як \ s відповідає усім пробілам Unicode. Це дивно, але всі сучасні браузери дотримуються специфікацій. У std :: regex всі скорочення є лише ASCII при використанні рядків char . У Visual C ++, але не в C ++ Builder, вони підтримують Unicode при використанні рядків wchar_t . Класи POSIX також відповідають символам, що не належать до ASCII, при використанні wchar_t у Visual C ++, але не включають послідовно всі символи Unicode, які можна було б очікувати.

На практиці в основному використовується граматика ECMAScript. Це граматика за замовчуванням і пропонує набагато більше можливостей, ніж інші граматики.

Що ж треба для створення об’єкта регулярного виразу?

Перш ніж використовувати регулярний вираз, потрібно створити об’єкт класу шаблону std :: basic_regex. Ви можете легко зробити це за допомогою екземпляра std :: regex цього класу шаблону, якщо ваша тема - це масив символів або об’єкт std :: string. Використовуйте екземпляр std :: wregex, якщо ваша тема - це масив wchar_t об’єкта std :: wstring.

Передайте свій регулярний вираз як рядок як перший параметр конструктору. Якщо ви хочете використовувати аромат регулярного виразу, відмінний від ECMAScript, передайте відповідну константу як другий параметр. Ви можете "або" цю константу за допомогою std :: regex_constants :: icase, щоб зробити регістр регулярних виразів нечутливим. Ви також можете "або" зробити це за допомогою std :: regex_constants :: nosubs, щоб перетворити всі групи захоплення в групи, які не захоплюють, що робить ваш регулярний вираз ефективнішим,

якщо ви дбаєте лише про загальний збіг регулярного виразу і не хочете витягувати відповідний текст будь-якою з груп захоплення.

Зробіть виклик std :: regex_search () із введеним рядком, як першим параметром, а об’єкт регулярних виразів - як другим параметром, щоб перевірити, чи може ваш регулярний вираз відповідати будь-якій частині рядка. Зробіть виклик std :: regex_match () з однаковими параметрами, якщо ви хочете перевірити, чи може ваш регулярний вираз відповідати цілому рядку. Оскільки std :: regex не має флагів, які збігаються виключно на початку та в кінці рядка, вам доведеться викликати regex_match (), коли використовується регулярний вираз для перевірки введення даних користувача.

І regex_search (), і regex_match () повертають лише true або false. Щоб отримати частину рядка, що відповідає regex_search (), або отримати частини рядка, що відповідають захопленню груп при використанні будь-якої функції, вам потрібно передати об’єкт класу шаблону std :: match_results як другий параметр. Тоді об’єкт регулярного виразу стає третім параметром. Створіть цей об'єкт, використовуючи конструктор за замовчуванням одного з цих чотирьох екземплярів шаблону:

std :: cmatch, коли ваша тема - це масив char

std :: smatch, коли ваша тема - це об'єкт std :: string

std :: wcmatch, коли ваша тема - це масив wchar_t

std :: wsmatch, коли ваш об'єкт є об'єктом std :: wstring

Коли виклик функції повертає true, ви можете викликати функції члена str (), position () і length () об'єкта match_results, щоб отримати відповідний текст, або початкову позицію та її довжину відповідності щодо рядок теми. Викличте ці функції-члени без параметра або з параметром 0, щоб отримати загальний збіг регулярного виразу. Зробіть виклик їх, що пройшли 1 або більше, щоб отримати збіг певної групи захоплення. Функція size () вказує кількість груп захоплення плюс одна для загального збігу. Таким чином, ви можете передати значення до size () - 1 іншим трьом функціям-членам.

Склавши все це разом, ми можемо отримати текст, відповідний першій групі захоплення, так:

```
std::string subject("Name: John Doe");
std::string result;
try {
  std::regex re("Name: (.*)");
  std::smatch match;
  if (std::regex_search(subject, match, re) && match.size() > 1) {
    result = match.str(1); //або result = match[1];
  } else {
    result = std::string("");
  }
} catch (std::regex_error& e) {
  // Syntax error in the regular expression
}
```

Щоб знайти ж всі збіги регулярних виразів у рядку, вам потрібно використовувати ітератор. Побудуйте об'єкт класу шаблону std :: regex_iterator, використовуючи один із цих чотирьох екземплярів шаблону:

std :: cregex_iterator, коли ваша тема - це масив char

std :: sregex_iterator, коли ваша тема - це об'єкт std :: string

std :: wcregex_iterator, коли ваша тема - це масив wchar_t

std :: wsregex_iterator, коли ваш предмет є об'єктом std :: wstring

Побудуйте один об’єкт, викликаючи конструктор із трьома параметрами: ітератор рядка, що вказує початкову позицію пошуку, ітератор рядка, що вказує кінцеву позицію пошуку, та об’єкт регулярного виразу. Якщо знайдено якісь збіги, об’єкт проведе перший збіг при побудові. Побудуйте інший об'єкт ітератора, використовуючи конструктор за замовчуванням, щоб отримати ітератор кінця послідовності. Ви можете порівняти перший об’єкт з другим, щоб визначити, чи є подальші збіги. Поки перший об'єкт не дорівнює другому, ви можете розмежувати перший об'єкт, щоб отримати об'єкт match_results.

```
std::string subject("This is a test");
try {
```

```
    std::regex re("\\w+");
    std::sregex_iterator next(subject.begin(), subject.end(), re);
    std::sregex_iterator end;
    while (next != end) {
      std::smatch match = *next;
      std::cout << match.str() << "\n";
      next++;
    }
  } catch (std::regex_error& e) {
    // Syntax error in the regular expression
  }
```

Щоб замінити всі збіги в рядку, зателефонуйте std :: regex_replace () з першим параметром вашої тематичної рядка, другим параметром - об’єктом регулярних виразів, а як третім - рядком із текстом заміни. Функція повертає новий рядок із застосованими замінами.

При використанні регулярних виразів радять користуйтеся інструментом перевірки регулярних виразів. Це значно спростить життя та передбачить багато нікому не потрібних багів, що можуть зламати весь код програми. Для перевірки виразів існує багато інструментів. які доступні онлайн у вільному доступі.

Потрібно на увазі, що надмірне використання регулярних виразів дає надмірне відчуття власної кмітливості. І це відмінний спосіб розсердити на вас ваших колег (і всіх, кому потрібно працювати з вашим кодом). Крім того, регулярні вирази є зайвими для більшості завдань синтаксичного аналізу, з якими доводиться працювати повсякденній роботі.

Регулярні вирази дійсно підходять для складних завдань, де рукописний код синтаксичного аналізу в будь-якому випадку буде настільки ж повільним і для надзвичайно простих завдань, коли читабельність і надійність регулярних виразів переважують їхні витрати на продуктивність.