На рисунке 2 – снимок первой в истории вебкамеры, которая была запущена в 1991 году и показывала кофеварку в Троянской комнате Кембриджского университета. На сегодняшний день она не работает, поскольку была отключена 22 августа 2001 года.

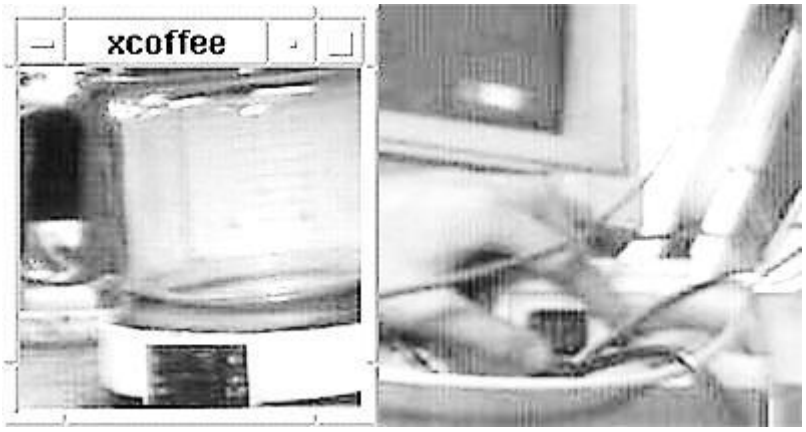

Рис. 2. Первый в истории снимок с вебкамеры

Веб камера играет важную роль в повседневной жизни большинства жителей планеты. Данное устройство позволяет обмениваться изображениями, видеозаписями как внутри сети, так и в инетрнете.

Также используется для наблюдения за улицами, охраны помещений, устанавливают в заповедниках для наблюдение за редкими видами животных, на спутники, космические станции, такие как Международная космическая станция.

Веб камера может быть задействована в таких программах, как, например: Skype, Viber, WhatsApp.

Список использованных источников:

1. Авдеев. В. Периферийные устройства: интерфейсы, схемотехника, программирование / В. Авдеев. – М.: ДМК Пресс, 2009. – 848 с.

2. Свободная энциклопедия, 2016 – Режим доступа: https://wikipedia.org

Гетьманенко О.В.
студент,
Науковий керівник: **Гавриленко О.В.**
доцент,
Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»

СТВОРЕННЯ ПАРСЕР-КОМБІНАТОРІВ З ПОДАЛЬШОЮ ОПТИМІЗАЦІЄЮ

Переважна більшість коду для сучасного програмного забезпечення створюється за об’єктно-орієнтовним стилем програмування. Проте у деяких

випадках використання функціонального стилю є невід'ємною частиною для досягнення успішного результату як з боку зручності застосування, так і швидкодії.

Як створюються складні системи? Кожну систему можна розділити на окремі підсистеми, а підсистеми – на окремі функціональні блоки. Аналогічно зі створенням коду: спочатку пишуться прості блоки, які об'єднуються у більш складні, створені групи, потім складаються у ще одні і врешті-решт – програма. Це не що інше як опис парадигми «розділяй і володарюй».

В основі процедурного стилю лежить опис процедур, які викликають інші процедури і разом змінюють загальні дані. Об'єктно-орієнтований стиль працює з більш складними структурами даних, шаблони для яких описані в класах. Функціональний стиль використовує композицію (з'єднання) функцій.

Чим же відрізняється композиція функцій від композиції процедур і об'єктів? Основа функціонального підходу – чистота функцій, це означає, що результат роботи функцій залежить тільки від вхідних параметрів. Саме функції, що приймають та/або повертають в якості результату інші функції, називаються функціями вищого порядку. Саме вони є зручними для створення парсер-комбінаторів.

Що ж являє собою парсер? Для спрощення парсером будемо вважати функцію, яка аналізує рядок (здійснює пошук даних за певним правилом) і в разі успіху повертає пару значень: результат розбору і залишок рядка. Парсер-комбінатор – це широковідома техніка створення парсерів – функцій розбору рядків (чи інших послідовностей даних) на знаходження в них даних з урахуванням граматики. Прикладом може бути сумування всіх чисел у текстовому рядку без використання стандартних функцій.

Функціональні мови дозволяють будувати парсери динамічно, використовуючи прості функції і комбінатори для синтезу за правилами граматики складних парсерів із простих.

Класичним варіантом буде функція, яка отримує на вхід список нерозібраних символів і в разі успіху повертає потрібне значення і залишок списку, що не був проаналізований, а у випадку невдачі повертає хибний результат, залишаючи список недоторканим.

Кожна така функція є «цеглиною» для побудови парсер-комбінатора.

Наприклад, якщо у нас є парсери і ми хочемо застосувати кожен з них до рядка, то можемо визначити комбінатор парсерів – функцію, яка створює новий аналізатор на основі існуючих. Це буде функція, яка приймає результат поточного парсера і повертає на його основі новий парсер, який продовжує розбір рядка з того місця, де зупинився попередній.

Таким чином ми скомбінували і отримали досить складний парсер, який спочатку аналізує лише один символ, за ним ще два і повертає результат аналізу. Процес роботи парсер-комбінаторів складається з конструювання замикань, викликів функцій і виділення пам'яті для проміжних результатів, що створює навантаження і погіршує швидкодію.

Якщо розглянути парсер, який розбирає один символ, то побачимо, що його можна описати простим кінцевим автоматом (КА) з наступним графом станів:

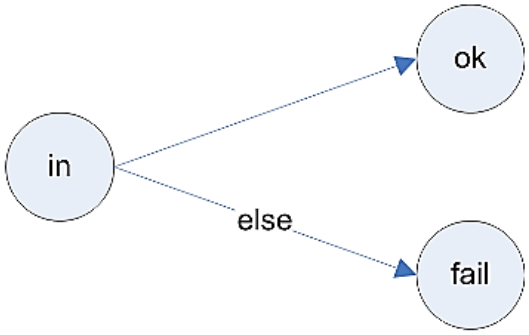

Рис. 1. Загальна схема простого кінцевого автомата

З початку він знаходиться в стані **in**, якщо символ підходить (результат розбору), то він переходить в стан **ok**, повертаючи успішний результат і залишок списку, а якщо символ не підходить, то автомат переходить в стан **fail**, не змінюючи при цьому список, який йому передавався. Загальна характеристика парсера полягає в тому, що він має один вхідний стан і два вихідних – успіх чи невдачу (поразку).

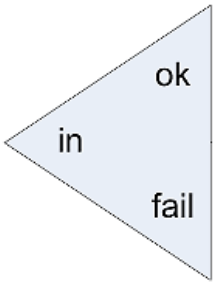

Рис. 2. Загальна схема парсера

Отож при з'єднані кількох простих автоматів можна отримати складний парсер-комбінатор і побудувати відповідний граф. Результат залишатиметься завжди однаковим – один вхідний стан і два вихідних, тому підходить для подальшого маніпулювання іншими комбінаторами.

Приклад об'єднання двох парсерів.

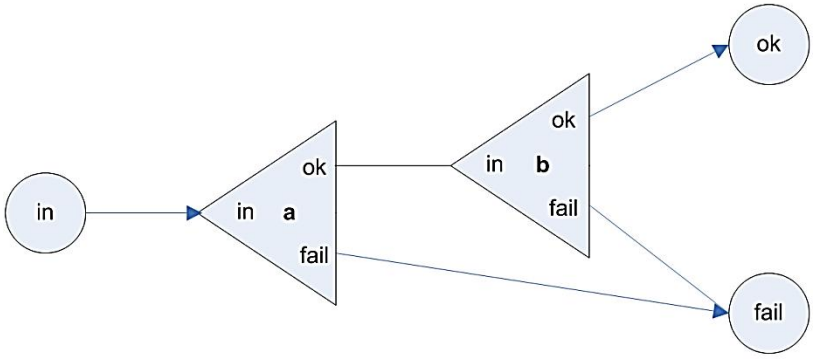

Рис. 3. Схема об'єднання двох парсерів (ab)

Створений ланцюг працює відповідно до логіки оператора послідовності, повертає успіх, якщо успішно по черзі спрацювали два парсери, із яких будувалася послідовність.

Аналогічно описуються комбінатори альтернативи.

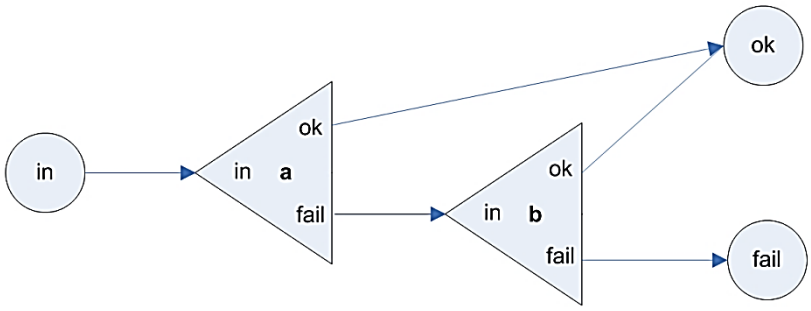

Рис. 4. Схема об'єднання автоматів для опису альтернативи (a|b)

Комбінуючи таким чином автомати, можна отримати ті ж парсери, що ми могли побудувати за допомогою класичних парсер-комбінаторів, окрім деяких рекурсивних. Аналогічно тому, як у класичних парсер-комбінаторах ми використовували декілька операторів і функцій для побудови складних парсерів із простих, тут опишемо набір аналогічних операторів і функцій, які будуть створювати граф кінцевого автомата із інших більш простих графів. Потім побудований граф може бути спрощений і перетворений у таблицю, по якій функція-парсер уже зможе працювати без надлишкових витрат.

Єдина складність – опис семантичних дій по створенню різних розібраних значень. У класичному підході використання замикання і розібрані раніше значення можна було використовувати пізніше для синтезу більш складних. Проте результатом комбінаторів була монолітна функція, яку перетворити і спростити вже не можна. Як варіант вирішення – семантичні дії передають один одному дані через динамічні змінні, які доступні їм всім.

Граф можна описати набором вершин, кожна із яких може мати декілька переходів із себе у інші вершини. У графі кінцевого автомата парсера є три види переходів:

1. Зустрівся потрібний символ.
2. Вітка else.
3. Безумовний перехід.

Кожний перехід може супроводжуватися якою-небудь подією. Подія переходу отримує на вхід символ у якості аргумента (позицію, з якої потрібно почати розбір), інші види переходів нічого не отримують і працюють виключно з описаним ззовні станом.

Перехід за вказаним символом змінює поточну позицію в тексті, що аналізується. Якщо якийсь із парсерів не зміг розібрати свою частину, як і у класичному варіанті, поточна позиція повинна залишитися тією ж, що була і до початку його роботи. Для цього ми повинні уміти її запам'ятовувати і при необхідності відновлювати. Так як складні парсери зазвичай складаються із вкладених один в одного більш простих, нам потрібен стек для зберігання позицій.

При вході в складний парсер поточна позиція буде запам'ятовуватися в стеку, при невдалому результаті вона буде вилучатися з нього і використовуватися для повернення, а при вдалому розвитку буде викидатися із стеку як більш не потрібна.

Основною метою статті є демонстрація того, як написання маленьких простих (і функціонально чистих) функцій і так само простих способів їх комбінування дозволяє будувати складні функції простим способом. А в простому й зрозумілому коді і помилок менше.

Список використаних джерел:

1. Комбинаторные парсеры. Часть 1 [Електронний ресурс] // Метавычисления и специализация программ. – 2010. – Режим доступу до ресурсу: http://metacomputation-ru.blogspot.com/2010/04/1.html

2. Попов Д. Оптимизирующие парсер-комбинаторы / Дмитрий Попов // Практика функционального программирования. – 2010. – № 5. – С. 66–74.

3. Функції вищих порядків і монади для PHP'шників [Електронний ресурс] // IT українською. – 2016. – Режим доступу до ресурсу: http://it-ua.info/news/2016/09/19/funkc-vischih-poryadkv-monadi-dlya-phpshnikv.html

4. Gerzic A. Write Your Own Regular Expression Parser [Електронний ресурс] / AmerGerzic // codeguru. – 2003. – Режим доступу до ресурсу: http://www.codeguru.com/cpp/cpp/cpp_mfc/parsing/article.php/c4093/Write-Your-Own-Regular-Expression-Parser.htm

Данилейко О.К.
старший викладач;
Бондаревський С.Л.
кандидат технічних наук, доцент;
Києвич Д.Ю.
студент,
ДВНЗ «Криворізький національний університет»

РОЗРОБКА СТЕНДА СВІТЛОФОРНОГО ОБ'ЄКТА НА БАЗІ ОБЛАДНЯННЯ ТОВ «ВО ОВЕН»

Правильна організація управлінням дорожнього руху дозволяє досягти умов, при яких зменшується ймовірність виникнення дорожньо-транспортних пригод, травм людей, нещасних випадків, заторів і виникнення нештатних ситуацій. Саме тому рішення цього питання має носити систематичний характер.

В Кривому розі з 2013 року почала діяти «Програма розвитку і безпеки дорожнього руху в Кривому Розі», згідно з якою в місті проводяться ряд заходів щодо покращення ситуації з безпекою дорожнього руху. Разом з цією програмою того ж року розпочала діяти іще одна програма «Автоматизована система управління світлофорних об'єктів» метою якої є дистанційне управління світлофорами через диспетчерську. Особливістю даної системи є те, що вона дозволяє створювати так звану «Зелену хвилю», тобто створити безперервний коридор для руху автомобілів через ряд світлофорів.

На базі лабораторії «Енергоефективні системи та технології в електромеханіці» кафедри електромеханіки ДВНЗ «Криворізький національний