UNIVERSIDAD DE GUADALAJARA

CENTRO UNIVERSITARIO DE CIENCIAS EXACTAS E INGENIERÍAS

DIVISIÓN DE ELECTRÓNICA Y COMPUTACIÓN

DEPARTAMENTO DE CIENCIAS COMPUTACIONALES

Ingeniería en computación

Traductores de Lenguajes II

SECCIÓN D-07

Tarea 7

Edgar Agustín Martinez González

CÓDIGO: 220286695

02/10/2022

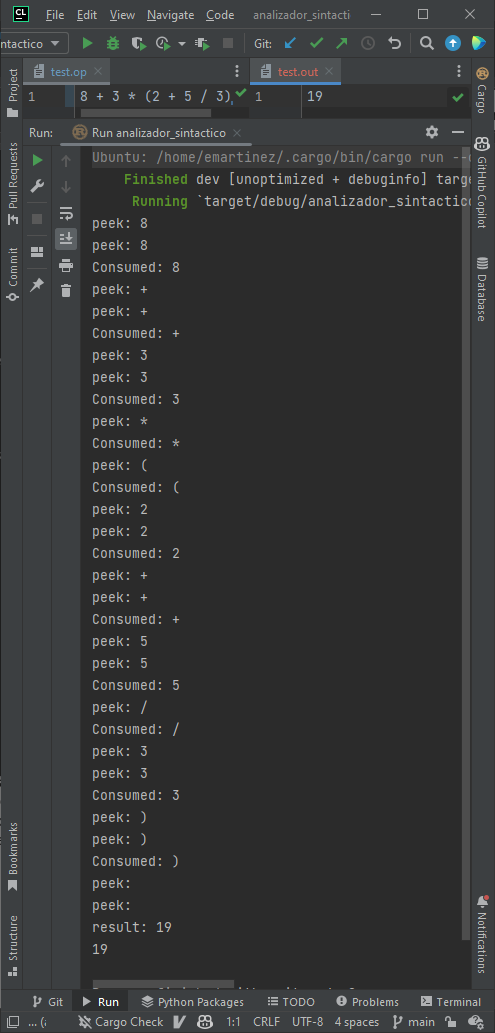
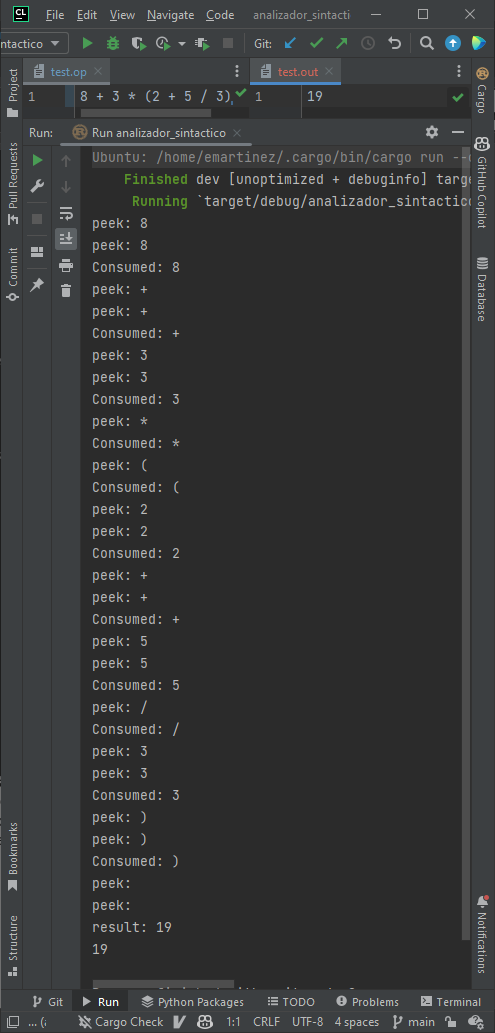
**Introducción**

Un analizador léxico lee los caracteres de la entrada y los agrupa en “objetos token”. Junto con un símbolo de terminal que se utiliza para las decisiones de análisis sintáctico, un objeto token lleva información adicional en forma de valores de atributos. Hasta ahora, no hemos tenido la necesidad de diferenciar entre los términos “token” y “terminal”, ya que el analizador sintáctico ignora los valores de los atributos que lleva un token. En esta sección, un token es un terminal junto con información adicional.

A una secuencia de caracteres de entrada que conforman un solo token se le conoce como lexema. Por ende, podemos decir que el analizador léxico aísla a un analizador sintáctico de la representación tipo lexema de los tokens.

**Pantallazos**

En el ejemplo se ve la operacion “8 + 3 \* (2 + 5 / 3) y el analisis de los tokens obtenidos del archivo fuente y con los que se obtiene de resultado 19.



**Conclusiones**

Los atributos semánticos y heredaros permiten mucho mas fácil ver el flujo de información como lo analiza la computadora desde llamadas a funciones recursivas y esto nos ayuda a planificar y diseñar desde antes las funciones de nuestro analizador sintáctico

**Codigo Fuente**

**use std::fs;**

**use std::fs::File;**

**use std::io::Read;**

**use std::io::Write;**

**const SOURCE\_FILE: &str = "example/test.op";**

**const TEMP\_FILE: &str = "example/tes.tmp";**

**const DEST\_FILE: &str = "example/test.res";**

**fn expr() -> i32 {**

**let term\_result = term();**

**expr\_tail( term\_result)**

**}**

**fn expr\_tail(inherited: i32) -> i32 {**

**match peek() {**

**'+' => {**

**consume('+');**

**expr\_tail( inherited + term())**

**}**

**'-' => {**

**consume('-');**

**expr\_tail( inherited - term())**

**}**

**\_ => inherited**

**}**

**}**

**fn term() -> i32 {**

**term\_tail(factor())**

**}**

**fn term\_tail( inherited: i32) -> i32 {**

**match peek() {**

**'\*' => {**

**consume('\*');**

**term\_tail( inherited \* factor())**

**}**

**'/' => {**

**consume('/');**

**term\_tail( inherited / factor())**

**}**

**\_ => inherited**

**}**

**}**

**fn factor() -> i32 {**

**match peek() {**

**'(' => {**

**consume('(');**

**let result = expr();**

**consume(')');**

**result**

**}**

**\_ => digit()**

**}**

**}**

**fn digit() -> i32 {**

**let peek = peek();**

**match peek {**

**'0' => {**

**consume('0');**

**0**

**},**

**'1' => {**

**consume('1');**

**1**

**},**

**'2' => {**

**consume('2');**

**2**

**},**

**'3' => {**

**consume('3');**

**3**

**},**

**'4' => {**

**consume('4');**

**4**

**},**

**'5' => {**

**consume('5');**

**5**

**},**

**'6' => {**

**consume('6');**

**6**

**},**

**'7' => {**

**consume('7');**

**7**

**},**

**'8' => {**

**consume('8');**

**8**

**},**

**'9' => {**

**consume('9');**

**9**

**},**

**\_ => panic!("Unexpected character: {}", peek)**

**}**

**}**

**fn initialize\_files() {**

**let mut file = File::open(SOURCE\_FILE).unwrap();**

**let mut contents = String::new();**

**file.read\_to\_string(&mut contents).unwrap();**

**let mut file = File::create(TEMP\_FILE).unwrap();**

**write!(file, "{}", contents).unwrap();**

**}**

**fn read\_temp\_file() -> String {**

**let mut file = File::open(TEMP\_FILE).unwrap();**

**let mut contents = String::new();**

**file.read\_to\_string(&mut contents).unwrap();**

**contents**

**}**

**fn consume(expected: char) {**

**let contents = read\_temp\_file();**

**let mut file = File::create(TEMP\_FILE).unwrap();**

**let chars = contents.chars();**

**let mut whitespaces = 0;**

**let consumed = chars.clone().skip\_while(|c| {**

**if c.is\_whitespace() {**

**whitespaces += 1;**

**true**

**} else {**

**false**

**}**

**}).next().unwrap();**

**println!("Consumed: {}", consumed);**

**if consumed != expected {**

**panic!("Syntax error: expected {}, found {}", expected, consumed);**

**}**

**let new\_contents = &contents[1 + whitespaces..];**

**write!(file, "{}", new\_contents).unwrap();**

**}**

**fn peek() -> char {**

**let contents = read\_temp\_file();**

**let chars = contents.chars();**

**let peek = chars.skip\_while(|c| c.is\_whitespace()).next().unwrap\_or(' ');**

**println!("peek: {}", peek);**

**peek**

**}**

**pub fn syntax\_analysis() ->i32 {**

**initialize\_files();**

**let result = expr();**

**println!("result: {}", result);**

**let mut dest\_file = File::create(DEST\_FILE).unwrap();**

**write!(dest\_file, "{}", result).unwrap();**

**fs::remove\_file(TEMP\_FILE);**

**result**

**}**