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

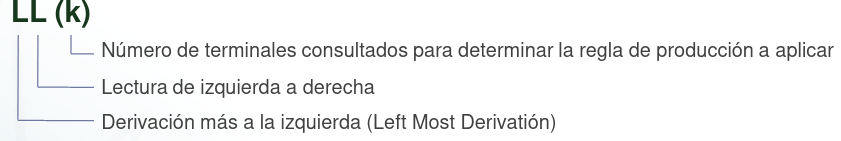
CÓDIGO: 220286695

26/09/2022

**Introducción**

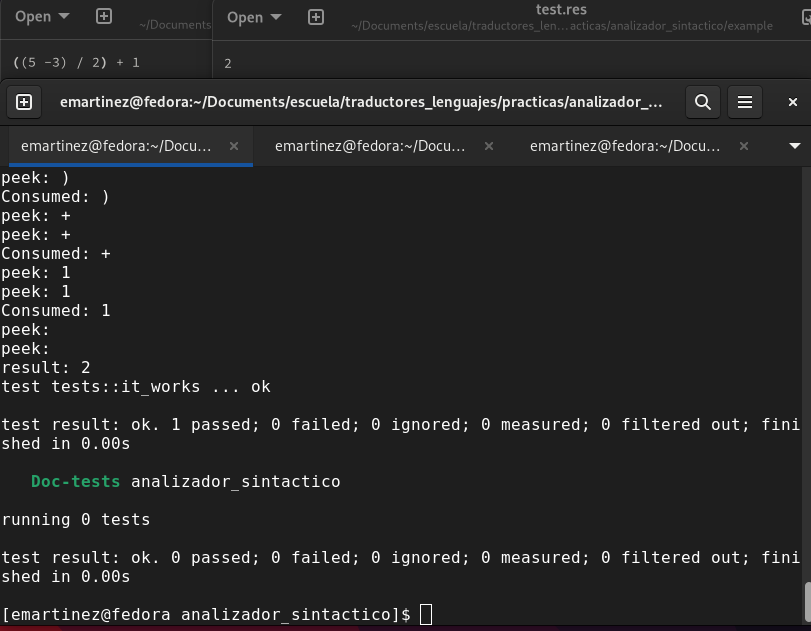
El parser (analizador sintáctico) utiliza los primeros componentes de los tokens producidos por el analizador de léxico para crear una representación intermedia en forma de árbol que describa la estructura gramatical del flujo de tokens. Una representación típica es el árbol sintáctico, en el cual cada nodo interior representa una operación y los hijos del nodo representan los argumentos de la operación.

Los analizadores sintácticos predictivos son capaces de decidir qué regla de producción aplicar a cada paso en función de los elementos terminales que se encuentran en la cabeza de lectura de la cadena de entrada. Como consecuencia se consigue un proceso de análisis con complejidad lineal O (n) con respecto al tamaño del problema. Estos analizadores son llamados genéricamente analizadores LL (K).

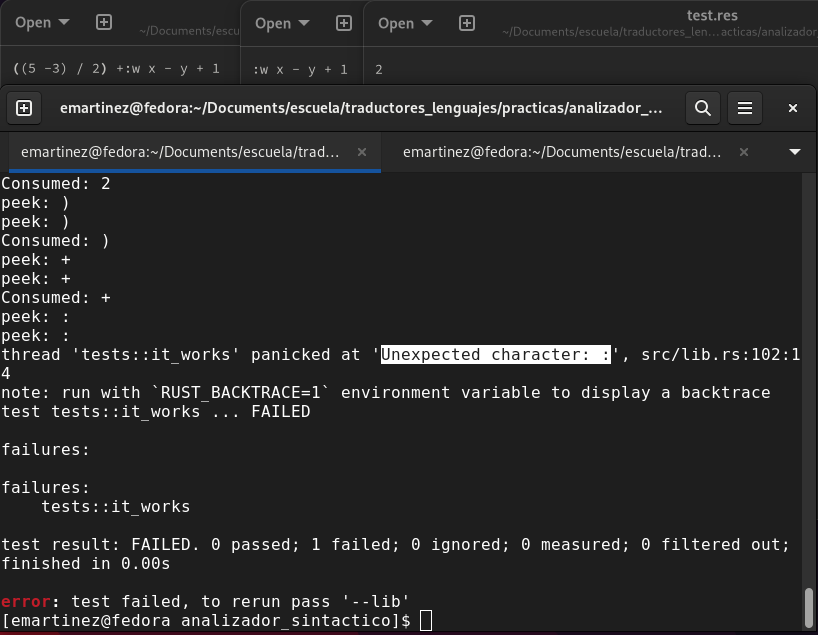


**Pantallazos**

En el primer caso el archivo de entrada contiene “((5 -3) / 2) + 1” y el archivo resultado da “2”



En el siguiente caso el archivo de entrada contiene “((5 -3) / 2) +:w x – y + 1” que es una cadena invalida. Aqui aun se puede observar el archivo temporal usado ya que al no terminar la compilación este no se elimino y quedo en el punto en que estaba antes de toparse con el error de sintaxis. El archivo resultado se quedo igual que en el caso pasado por que no llego al punto de guardado debido al error de compilacion.



**Conclusiones**

El analizador sintactico solamente es la parte que verifica que cada caracter sea correcto siguiendo la gramatica del lenguaje. Todo el trabajo hecho para que se obtenga el resultado no corresponde a esta practica pero no es trabajo en vano por que en la siguiente practica haremos uso de dicha funcion.

**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**

**}**