Área de Ingeniería En Computación

Compiladores e Intérpretes

I Avance

Gramática de un Lenguaje Loco

Estudiantes:

Ignacio Castillo Montero - 2021579556

Miguel Cubero Valverde - 2023271617

José Miguel González Barrantes 2023087564

Alejandro Gutierrez Chaves - 2023066266

Felipe Masís Calderón - 20230478825

Profesor:

Aurelio Sanabria Rodríguez

25 de Marzo

I Semestre, 2025

Tabla de Contenido

1. Motivación

Desde nuestros inicios en el mundillo de la programación siempre se nos ha explicado cómo funciona la programación con un enfoque no tan técnico; se nos dice que la programación consiste en seguir una serie de pasos para poder llegar a un producto nuevo y único creado por uno mismo. He aquí donde muchos profesores utilizaban una analogía muy curiosa, interesante pero que a lo largo de los años nos hemos dado cuenta que tiene mucho sentido; la cocina.

Muchos profesores utilizan hasta la fecha la idea de que la programación es el seguimiento de un algoritmo, y un algoritmo es como una “receta”. Esto fue lo que nos dió esa pequeña chispa cuando se nos vino a la mente la cocina como una idea central de un lenguaje de programación. Al instante nos dimos cuenta que era una excelente idea, aparte de que algunos del grupo son muy apasionados por la cocina, consideramos que nos permite pensar todo lo que conlleva la programación pero desde un enfoque completamente distinto pero a la misma vez tenga alguna forma de relación general entre ambos. Es bien sabido que una receta conlleva sus ingredientes, medidas, pasos a seguir, recomendaciones, controles de calidad, y otros muchos aspectos más. Pues si nos ponemos a pensar más a detalle, un lenguaje de programación también toma en cuenta aspectos como variables (ingredientes), métricas de eficiencia (mediciones), algoritmos (pasos), mejores prácticas (recomendaciones) y la calidad de un programa. Por lo que vimos como una oportunidad el poder implementar un lenguaje de programación sea fácil de entender y aplicar para todas las personas, desde aquellas que no tienen un conocimiento de lenguajes de programación muy amplio a personas más experimentadas. Adicionalmente consideramos que es realmente importante que todas las personas disfruten y se vean interesados en aprender nuestro lenguaje de programación, he aquí la importancia de que intentemos cumplir nuestros objetivo para así marcar la diferencia entre los demás lenguajes de programación.

Con todo esto en mente, nuestro objetivo es claro: diseñar un lenguaje de programación que sea accesible, intuitivo y que logre capturar la esencia de la cocina como una metáfora perfecta de la programación. Queremos que cualquiera, desde personas sin experiencia hasta programadores avanzados, pueda entender y aplicar sus conceptos sin mayor dificultad, logrando así un aprendizaje más fluido y natural.

Al igual que en la cocina, donde una receta bien escrita hace la diferencia entre un plato exitoso y un desastre culinario, en la programación una estructura clara y bien definida es clave para obtener un software eficiente. Es por esto que nos enfocamos en crear un lenguaje que no solo sea fácil de aprender, sino que también incentive las mejores prácticas sin necesidad de imponer reglas rígidas que limiten el poco conocimiento de este.

Sabemos que aprender a programar puede ser una experiencia frustrante para muchos, especialmente cuando se enfrentan a una sintaxis compleja o a conceptos abstractos difíciles de visualizar. Sin embargo, creemos que utilizando un enfoque más cercano y cotidiano como el de la cocina, podemos hacer que el proceso sea mucho más amigable y atractivo. Nuestra meta es que el aprendizaje de la programación se sienta como seguir una receta bien explicada, donde cada paso tenga sentido y el resultado final sea un producto único creado por el propio usuario.

En última instancia, buscamos marcar la diferencia con un lenguaje que no solo enseñe a programar, sino que también motive a las personas a experimentar, aprender y desarrollar su propio estilo, tal como sucede en la cocina. Si logramos despertar esa chispa de curiosidad y creatividad en quienes lo usen, sabremos que vamos por el camino correcto.

Origen de la temática del lenguaje: El cocinar no es tan distinto a programar por mas raro que suene, es una tarea que hay que pensar y crear algo, y hacerlo paso a paso, usualmente en procesos pequeños que un momento a otro se juntan para crear un objetivo final grande, por lo cual surgió esta temática.

Origen del nombre del lenguaje: Debido a que la temática del lenguaje está relacionada con la cocina y algo común y sencillo de hacer en la cocina es un sandwich a eso añadir que esta basado en python (el cual es relativamente fácil para iniciar) y luego el ensamblar, que es algo que se hace con un sándwich (o armar).

2. Puntos Fuertes y Chistosos del Lenguaje

La gramática y en sí la sintaxis del lenguaje tratan de implementar signos que hacen referencia a utensilios o acciones que uno realizaría en la cocina. En el caso de los comentarios, el lenguaje los implementa mediante un signo parecido a un tenedor “-E”, y para terminar el comentario existe el signo de “o-”, que hace referencia a una cuchara. Esto no solo le da una identidad única al lenguaje, sino que también lo diferencia de otros lenguajes como C, Java o Python, donde los comentarios se marcan con //, /* */ o #.

Además, se incluyó la regla de *quemar*, que representa una acción típica que todos hemos hecho alguna vez en la cocina, especialmente cuando somos novatos. Este concepto es análogo a los errores o excepciones en otros lenguajes, pero con una presentación más temática y fácil de relacionar. Mientras que en lenguajes como C un error de ejecución puede resultar en un *segmentation fault*, aquí se trata de una acción común en la cocina, lo que lo hace más intuitivo y menos técnico para quienes se acercan a la programación por primera vez.

Puntos fuertes: Una ventaja clave de este lenguaje es que está muy alineado con la forma de pensar que uno tendría al cocinar. Esto lo hace natural, intuitivo y no requiere un análisis profundo para comprender las distintas reglas y elementos que lo componen. En comparación con otros lenguajes, donde la sintaxis puede parecer abstracta y difícil de recordar, aquí todo tiene un significado lógico dentro del contexto de la cocina.

Puntos débiles: Sin embargo, este mismo enfoque puede ser una desventaja para quienes ya están acostumbrados a otros lenguajes de programación. La necesidad de aprender una sintaxis tan distinta puede generar una curva de aprendizaje innecesaria

para programadores experimentados. Además, la estructura basada en la cocina podría limitar su aplicación en proyectos más formales o en entornos profesionales donde la claridad y la compatibilidad con otros lenguajes son esenciales.

Aspectos chistosos: El lenguaje tiene un lado humorístico, ya que incorpora acciones y errores que todos hemos vivido al cocinar. Un ejemplo de esto es la regla de *quemar*, ya que es una manera divertida de representar fallos que en otros lenguajes se tratarían con mensajes de error fríos y técnicos. Aquí, la experiencia de programar se siente más cercana a la vida real y menos como una tarea estrictamente matemática o computacional.

3. Gramática EBNF

Nombre del lenguaje: Ensamblando un Sándwich

Tema para usar: Cocinar

Necesario ■ ■

Para ordenar la gramática ■

Símbolo inicial

Cocina := "{" (Chef | Receta | Reseña)+ "}"

El símbolo inicial Cocina representa el bloque principal del programa, que puede contener múltiples funciones (Chef), instrucciones (Receta) y comentarios (Reseña).

Definición de funciones: Chef

Chef := "michelin" Nombre "(" IngredienteLista? ")" "{" Receta+ Reseña* "}"

IngredienteLista := Ingrediente ("," Ingrediente)*

Un Chef representa la definición de una función. Utiliza la palabra clave michelin como declaración, seguida por el nombre de la función y una lista de parámetros (ingredientes). Cada cuerpo de la función debe incluir al menos una instrucción (Receta) y puede contener comentarios (Reseña).

Definición de variables: Ingrediente

Ingrediente := Cooking Nombre (Estado)?

Cooking := “int” | “string" | “bool” | “list” | “float”

Ingrediente son variables, Cooking reconoce números enteros, caracteres, números flotantes, booleanos o listas (el tipo de dato básicamente)

Instrucciones del lenguaje: Receta

Receta := Mezclar | Bifurcar | Emplatar | Error | Ajustar | Incorporar

Las Recetas son instrucciones ejecutables dentro de las funciones. Existen diferentes tipos de receta, que representan estructuras de control, operaciones, asignaciones y retorno.

Ciclos: Mezclar

Mezclar := "integrar" "(" Condicion ")" "{" Preparacion+ "}"

Mezclar representa una estructura de repetición (ciclo). Se ejecuta mientras la condición indicada se mantenga verdadera. Dentro del bloque se colocan una o más preparaciones (instrucciones).

Condicionales: Bifurcar

Bifurcar := "if" Ingrediente Comparacion Ingrediente Accion
("elif" Ingrediente Comparacion Ingrediente Accion)*
("else" Accion)?

Bifurcar permite el uso de condicionales. Acepta estructuras if, elif y else, comparando dos ingredientes mediante operadores relacionales.

Operaciones aritméticas: Ajustar

Ajustar := Numero Operador Numero
Operador := "batir" | "colar" | "amasar" | "partir" | "sobras"

Ajustar permite realizar operaciones aritméticas básicas como suma (batir), resta (colar), multiplicación (amasar), división (partir) y módulo (sobras).

Asignación de variables: Incorporar

Incorporar := Ingrediente "=" Valor

Valor := Numero | Texto | Crudo | Lista

Incorporar asigna un valor a un ingrediente. El valor puede ser un número, texto, valor booleano (Crudo) o una lista.

Declaración de error: Error

Error := "quemo" Texto

Error representa un estado de error dentro de una función. Su uso implica una terminación anómala del flujo de ejecución.

Mostrar valores: Emplatar

Emplatar := "servir" Nombre

Emplatar permite retornar un valor desde una función. Es el equivalente a una instrucción print.

Léxico (Tokens)

Usado de forma normal para escribir los textos según estándar de la EBNF

Texto := " \w+ \s+ "?

Formato para organizar números

Numero := (Entero | Flotante)

La forma de escribir números, pueden ser negativos con el “-” el “.” es necesario para los flotantes

Entero := (-)? [0-9]+

Flotante:= (-)? [0-9]+ (. [0-9]+)?

Nombre se usa para nombrar las variables ya que “patata1” entonces puede ser una mezcla

Nombre := “ \w+ \s+ \d+ “ ?

Crudo se usa para saber si algo es verdadero o falso

Crudo := True | False

Lista, se usa para definir listas

Lista := “[“ (\d* | \w* | \s* | Lista)* “]”

Comparaciones

Comparacion := (“mismo sabor que” | “mas sazonado que” | “menos cocido que” | “tan horneado como” | “tan dulce como”)

Mismo Sabor que es Igual, Más Sazonado que es Mayor que, Menos Cocido que es Menor que, Tan Horneado como es Mayor Igual que, Tan Dulce como es Menor o igual que

Operador := (batir | colar | amasar | partir | sobras)

Operadores aritméticos para estos casos en específico los operadores son: Batir es el equivalente a “+”, Colar es el equivalente a “-”, Amasar es el equivalente a “*”, Partir es el equivalente a “/” y Sobras representa “%”

Funciones auxiliares

Reseña := "-E" Texto "-o"

Funciones auxiliares

Ambiente estandar

Pelar; lo que hace es que se le pasa una variable(ingrediente) y conforme se usa, el segundo valor(Numero) define que tanto se le va restando

Pelar := "pelar" "(" Nombre "," Numero ")"

Marinar; Variables que cambian su respectivo valor con el tiempo, no tiene que ser necesariamente cambio de valor, sino de propiedades

Marinar := "marinar" "(" Nombre "," (Numero | Texto) ")"

4. Ejemplos del Código

Ejemplo 1: Función de sumatoria

```
{
  michelin sumatoria(int limite) {
    incorporar int suma = 0
    incorporar int i = 1

    integrar(i menos cocido que limite) {
      ajustar suma = suma batir i
      ajustar i = i batir 1
    }

    servir "La suma es: "
```

```
    servir suma
}

  -E Esta receta suma los números desde 1 hasta el límite dado -o
}
```

Ejemplo 2: Serie de Fibonacci

```
{
  michelin fibonacci(int n) {
    incorporar int a = 0
    incorporar int b = 1
    incorporar int contador = 0

    integrar(contador menos cocido que n) {
      servir "Número: "
      servir a

      incorporar int temp = a
      ajustar a = b
      ajustar b = temp batir b
      ajustar contador = contador batir 1
    }

    -E Esta receta imprime la serie de Fibonacci hasta el enésimo valor -o
  }
}
```

Ejemplo 3: Operaciones con listas y loops

```
{
  michelin sumar_lista(list ingredientes) {
```

```
    incorporar int suma = 0
    incorporar int i = 0

    integrar(i menos cocido que ingredientes.longitud) {
      ajustar suma = suma batir ingredientes[i]
      ajustar i = i batir 1
    }

    servir "La suma total es:"
    servir suma
  }

  -E Esta receta recorre una lista de ingredientes y devuelve la suma total -o
}
```

Ejemplo 4: Manipulación de strings

```
{
        Michelin Saludar(string nombre) {
        Incorporar saludo = "¡Hola " batir nombre batir "!"
        Emplatar Servir saludo
        }

        -E Función de demostración para strings -o
        Saludar("Master Chef")
}
```

Ejemplo 5: Operaciones matemáticas complejas

```
{
  michelin area_circulo(float radio) {
    incorporar float pi = 3.1416
    incorporar float area = 0

    ajustar area = pi amasar radio amasar radio

    servir "El área del círculo es:"
    servir area
  }

  -E Esta receta calcula el área de un círculo usando el radio proporcionado -o
}
```