MiniGo Compiler & Editor

Aaron González y Daniel Porras

ITCR

Compiladores e Intérpretes

Óscar Víquiez

3 de abril de 2024

MiniGo Compiler & Editor

Índice

Introducción

Este trabajo presenta un compilador desarrollado con ANTLR-4 para un lenguaje de programación inspirado en Go, pero con una sintaxis simplificada. Se describirá la gramática del lenguaje, la implementación del compilador y su funcionamiento.

Desarrollo

Soluciones e Implementación

El desarrollo del compilador se basa en el uso de ANTLR-4, una herramienta ampliamente utilizada para generar analizadores léxicos y sintácticos a partir de gramáticas formales. En este proyecto, se ha definido una gramática específica para el lenguaje simplificado inspirado en Go. Esta gramática describe la estructura sintáctica del lenguaje, incluyendo reglas para declaraciones, expresiones, estructuras de control y otros elementos fundamentales.

Una vez definida la gramática en ANTLR-4, se utiliza esta herramienta para generar el código fuente de un analizador léxico y sintáctico. Este código, escrito en Go, constituye el núcleo del compilador. El programa resultante recibe como entrada el nombre de un archivo que contiene el código fuente en el lenguaje simplificado. El analizador léxico y sintáctico se encarga de leer este archivo, tokenizar el código y verificar su estructura sintáctica según la gramática definida.

Durante el proceso de análisis, el compilador identifica posibles errores sintácticos, como declaraciones mal formadas o estructuras de control incorrectas. Si se encuentra algún error, el compilador muestra un mensaje indicando la naturaleza del problema y la ubicación aproximada en el código fuente. Por otro lado, si no se encuentran errores, el compilador devuelve un código de éxito, indicando que el código fuente es válido según la gramática definida.

Además del compilador en Go generado por ANTLR-4, se ha desarrollado un editor

de código con una interfaz gráfica amigable. Este editor es responsable de ejecutar el compilador y mostrar la salida al usuario.

Cuando el usuario abre un archivo en el editor, este envía el código al compilador para su análisis. El compilador procesa el código y, si encuentra errores, devuelve mensajes detallados sobre la naturaleza de los problemas encontrados y su ubicación en el código fuente.

El editor recibe esta salida del compilador y la muestra de manera clara y organizada al usuario, resaltando los errores y facilitando su comprensión.

Análisis de la gramática

La gramática define un lenguaje parecido a Go pero con simplificaciones. Define la estructura básica de un programa con un paquete, un identificador y una lista de declaraciones principales. Permite múltiples declaraciones de variables, tipos y funciones en cualquier orden. Las declaraciones de variables pueden ser simples, múltiples dentro de paréntesis o vacías. Las declaraciones de tipos pueden ser simples, múltiples dentro de paréntesis o vacías. Las funciones se declaran con un nombre, argumentos entre paréntesis y un tipo de retorno opcional. Las expresiones pueden ser simples, binarias, unarias o llamadas a función. Las expresiones primarias incluyen operandos, expresiones con selectores, índices, argumentos, append, length o capacidad. Los operandos pueden ser literales, identificadores o expresiones entre paréntesis. Las sentencias incluyen impresión, retorno, ruptura, continuación, sentencias simples, bloques de código, selección, condicionales y bucles.

Comparaciones

A continuación se muestran comparaciones de código escrito en `Go` y en `MiniGo`.

Retorno de múltiples variables

Go

```go
import (
    "fmt"
)

func div(numerador, denominador int) (int, error) {
    if denominador == 0 {
        return 0, fmt.Errorf("no se puede dividir entre cero")
    }

    return numerador/denominador
}

func main() {
    resultado, err := div(3, 0)
    if err != nil {
        panic(err)
    }
}
```

MiniGo

```
var err string;

func div(numerador, denominador int) int {
    if denominador == 0 {
        err = "no se puede dividir entre cero";
        return 0;
```

```
    };

    return numerador/denominador;
};

func main() {
    resultado := div(3, 0);
    if err != "" {
        panic(err);
    };
};
```

Cosas que no se pueden hacer en MiniGo

El siguiente código que sería 'válido'[1] hasta cierto punto en go, no es código válido de MiniGo.

```
type Persona struct {
    Nombre string;
    Edad int;
};

func NewPersona(nombre string, edad int) *Persona {
    return &Persona{
        Nombre: nombre,
        Edad: edad,
    };
};
```

[1] En go no es requerido el uso de punto y coma, el lenguaje no lo enfuerza, sino que este lo desalienta.

12

En MiniGo no existe el concepto de argumentos variádicos. El siguiente código causará un error.

```
func saludarATodos(nombres ...string) {
    for idx := 0; idx < len(nombres); idx++ {
        println("Hola", nombres[idx]);
    };
};
```

Para ver casos de prueba más extensos por favor referirse a las pruebas dentro del repositorio del compilador

Cuadro Explicativo

A continuación se encuentra un sumario de los requerimientos y su estado en el proyecto.

Requerimiento	Estado
Reconocimiento de tokens	Realizado
Manejo de comentarios	Realizado
Archivos de prueba	Realizado
Validación sintáctica	Realizado
Mostrar errores sintácticos (editor)	Realizado
Mostrar línea y columna de errores (editor)	Realizado

Conclusiones

Basándonos en el análisis exhaustivo de la gramática presentada, se puede concluir que el lenguaje propuesto representa una versión simplificada y más accesible del paradigma de programación de Go. Si bien mantiene muchos de los principios fundamentales de Go,

como la fuerte tipificación de variables, las funciones y las estructuras de control, se han realizado simplificaciones significativas en algunas áreas para mejorar la legibilidad y reducir la complejidad sintáctica.Estas modificaciones podrían resultar beneficiosas para aquellos nuevos en la programación o para proyectos que no requieran toda la complejidad y el rigor de Go en su forma tradicional. Sin embargo, es importante tener en cuenta que estas simplificaciones también pueden limitar la capacidad del lenguaje para abordar problemas más complejos y sofisticados de manera eficiente.