MiniGo: Compilador y Editor: Parte 3

Aaron González y Daniel Porras

ITCR

Compiladores e Intérpretes

Óscar Víquez

1 de junio de 2024

MiniGo: Compilador y Editor: Parte 3

Índice

Resultados

Punto	Estado
Declaraciones y usos de variables de tipos simples tanto globales como locales en asignaciones y expresiones.	Completo
Declaraciones y usos de arreglos de enteros. La forma más simple de declaración tipada.	Completo
Declaraciones y usos de métodos (procedimientos y funciones)	Completo
Instrucciones de control de flujo solamente IFs y LOOPs, sin uso de “break” ni “continue”.	Completo
Implementación de la función predefinida “println” para imprimir cualquier tipo simple y la función “len” para obtener el largo de un arreglo de enteros.	Completo
Expresiones simples o complejas, con o sin agrupación usando paréntesis para operaciones con los tipos de datos mencionados utilizando los operadores matemáticos y operadores lógicos.	Completo
Se permitirá el uso de cadenas RawString solamente y para impresiones dentro del “println”.	Completo

Características

1. Se pueden declarar variables de los 3 métodos que existen en go (typed, untyped & walrus).
2. Se pueden crear arreglos de cualquier tipo de datos, no solo enteros.
3. Se pueden crear funciones.
4. Se implementaron instrucciones de control de flujo, en este caso todas las variantes de if y todas las variantes de for, con el extra de que se implementaron los break statements.
5. Se implementó la función `println` esta puede imprimir una amplia variedad de tipos primitivos.
6. Se implementaron operaciones aritméticas básicas para `int` y `float`, para `strings` solamente se implementó concatenación. Además se implementaron todas las operaciones lógicas de comparación para `float` e `int`.
7. Se implementaron todos los tipos de string válidos (RawString & InterpretedString)

Pruebas

Manual de pruebas

En el repositorio minigo se encuentra una carpeta con todas las pruebas que se realizaron para asegurar el funcionamiento del compilador, solo es necesario compilarlas y correr los ejecutables resultantes para comprobar su funcionamiento.

Para utilizar el compilador se puede usar directamente desde el editor o como un programa standalone, primero de debe compilar el compilador usando el siguiente comando:

```
go build -o minigo ./cmd
```

> Es importante mantener el `./` (o `.\` si se encuentra en windows), de lo contrario se compilará el paquete cmd interno de go y no lo que se espera, este comando se debe correr desde la raíz del proyecto.

Uso del compilador

El compilador cuenta con dos comandos `build` y `run`, se utilizan de la siguiente manera:

```
./minigo build <archivo a compilar>
```

```
./minigo run <archivo a compilar y correr>
```

Esto producirá un error o un archivo ejecutable, si el archivo de código fuente se llamaba `main.minigo` entonces el ejecutable se llamará `main` (en linux) o `main.exe` (en windows).

El comando run lo único que hace diferente es que además de compilarlo ejecuta inmediatamente el archivo ejecutable producido, los archivos binarios se guardan directamente en la carpeta actual y por el momento no hemos agregado una opción para cambiar el nombre del archivo de salida.

Pruebas puntuales

Función seno

A continuación se muestra una implementación de la aproximación de Bhaskara (2011) a la función seno

```
1 package main;
2
3 var PI = 3.141592653589793;
4
5 func sin(theta float) float {
6     return (16.0 * theta * (PI - theta))/(5.0*PI*PI - (4.0*theta*(PI-theta)));
7 };
```

Secuencia de fibonacci

A continuación se muestra una implementación de una función que calcula el enésimo término de la secuencia de fibonacci

```
func Fibonacci(n int) int {

    if n == 0 {
        return 0;
    };

    a := 0;
    b := 1;

    c := 0;
    for i := 2; i <= n; i = i + 1 {
        c = a + b;
        a = b;
        b = c;
    };
    return c;
};
```

Función factorial

A continuación se muestra una implementación de la función factorial en minigo (n!)

```

func Factorial(n int) int {
    fact := 1;
    for i := 1; i <= n; i = i + 1 {
```

```
        fact *= i;
    };
    return fact;
};
```

Bubble sort

A continuación se muestra una implementación del algoritmo de ordenamiento `bubble sort`.

```

func BubbleSort() {
    var arr [5]int;
    arr[0] = 5;
    arr[1] = 3;
    arr[2] = 2;
    arr[3] = 4;
    arr[4] = 1;

    for i := 0; i < len(arr); i++ {
        printf("bubble sort before: %lld\n", arr[i]);
    };

    for i := 0; i < len(arr)-1; i = i + 1 {
        for j := 0; j < len(arr)-i-1; j = j + 1 {
            if arr[j] > arr[j+1] {
                temp := arr[j];
                arr[j] = arr[j+1];
                arr[j+1] = temp;
```

```
            };
        };
    };

    for i := 0; i < len(arr); i++ {
        printf("bubble sort after: %lld\n", arr[i]);
    };
};
```

Conclusión

El desarrollo de un lenguaje de programación es un proceso complejo que requiere una profunda comprensión de los fundamentos de la informática y la programación. A lo largo de este proyecto, se logró diseñar e implementar un lenguaje con soporte para tipos básicos, arreglos dimensionados y estructuras de control esenciales.

El proyecto permitió explorar aspectos clave del diseño de lenguajes, como la sintaxis, la semántica y la gestión de memoria. Además, se adquirió experiencia en la implementación de compiladores e intérpretes, así como en la resolución de problemas comunes en el desarrollo de lenguajes.

Bibliografía

Shirali, Shailesh A. 2011. «The bhaskara-aryabhata approximation to the sine function». *Mathematics Magazine* 84 (2): 98-107.