Umbra

Especificación del Lenguaje

1. *Tipado Estático.*

El lenguaje requiere que todas las variables y funciones sean declaradas explícitamente con su tipo. Esto incluye tipos básicos como int, float, bool, char, string, y array.

Ejemplo de declaración de variables:

int contador = 10

float tasa = 3.14

bool es_valido = true

2. *Declaración y Definición de Funciones*

Las funciones deben especificar su tipo de retorno mediante la sintaxis ->. Los parámetros también deben estar tipados explícitamente.

La sintaxis para la definición de funciones es la siguiente:

```
func nombre_funcion(tipo parametro1, tipo parametro2) -> tipo_retorno {
        // Cuerpo de la función
}
```

Ejemplo de función:

```
func suma(int a, int b) -> int {
        return a + b
}
```

3. *Manejo de Memoria*

El lenguaje utiliza las palabras clave new y delete para la asignación y liberación de memoria, respectivamente. Esto permite a los usuarios manejar explícitamente los recursos de memoria de manera intuitiva.

Ejemplo:

```
integer[] numeros = new int[10]

delete numeros
```

Punteros: El lenguaje utiliza punteros de estilo C/C++, proporcionando a los desarrolladores un control directo sobre la memoria.

Ejemplo:

Los punteros se usarán a través de la palabra reservada “ptr” y las referencias indicadas con la palabra reservada “ref”. Para acceder al dato al que apunta el puntero, se usa la palabra reserva access.

```
int a = 10;

ptr int ptr_a = ref a;

access a = 21;
```

esto para mejorar la legibilidad y facilitar el aprendizaje de los punteros usando nemotécnicos en lugar de símbolos.

4. *Operadores*

Se priorizan los operadores textuales equal, different, and, or sobre los operadores simbólicos tradicionales. Esto mejora la claridad y la legibilidad del código, especialmente para aquellos nuevos en la programación.

Operadores soportados:

Igualdad: equal (equivalente a ==)

Diferencia: different (equivalente a !=)

Lógicos: and, or (equivalente a &&, ||)

Relacionales: >, <, <=, >=

Ejemplo:

```
if (a equal b) and (c different d) {
        // Código
}
```

5. *Estructuras de Control*

Condicionales: El lenguaje soporta estructuras condicionales como if, else if, y else para la toma de decisiones, la condición puede o no ir con parentesis.

```
if condicion {
        // Código si la condición es verdadera
} else if otra_condicion {
        // Código si la otra condición es verdadera
} else {
        // Código si ninguna condición es verdadera
}
```

Bucles:

repeat x times: Ejecuta un bloque de código un número determinado de veces.

```
repeat n times {
        // Código que se ejecuta 5 veces
}
```

repeat if: Ejecuta un bloque de código mientras se cumpla una condición.

```
repeat if condicion {
        // Código que se ejecuta mientras la condición sea verdadera
}
```

6. *Agrupación de Código y Reglas de Indentación*

Los bloques de código se agrupan utilizando llaves {}, y la indentación es obligatoria. Esto garantiza una estructura clara y legible del código.

Ejemplo:

```
func ejemplo() -> void {
        if condicion {
                // Código
                repeat 3 times {
                        // Código repetido
                }
        }
}
```

7. *Manejo de Errores*

Las funciones pueden devolver valores especiales en caso de errores, y se recomienda el uso de mensajes de error explícitos para facilitar el manejo de fallos en tiempo de ejecución.

Ejemplo:

```
func factorial(int n) -> int {
        if n less_than 0 {
                print("Error: El número debe ser no negativo.");
                return -1;
        }
        // Código para calcular factorial
}
```

8. *Entrada y Salida*

El lenguaje proporciona funciones básicas para la entrada y salida de datos. print se utiliza para mostrar información en la consola, y input para capturar datos del usuario.

Ejemplo:

```
print("Ingrese un número:");
int num = input();
```

Ejemplo de Programa Completo

```
func factorial(int n) -> int {
        if n less_than 0 {
                print("Error: El número debe ser no negativo.");
                return -1;
        }
        int result = 1;
        repeat n times {
                result *= n;
                n -= 1;
        }
        return result;
}
func main() -> void {
        int[] arr = new int[10];
        if arr equal null {
                print("Error: No se pudo asignar memoria.");
                return;
        }
        int i = 0;
        repeat if i less_than 10 {
                arr[i] = factorial(i);
                print("Factorial de ", i, " es ", arr[i]);
```

```
            i += 1;
        }
        delete arr;
}
```