Diseño y desarrollo de un lenguaje de programación

Tema: Especificación lexica

Estudiante	Escuela	Asignatura
Rodrigo Infanzón Acosta rinfanzona@ulasalle.edu.pe	Carrera Profesional de Ingeniería de Software	Compiladores

Informe	Tema	Duración
01	Especificación lexica	06 horas

Semestre académico	Fecha de inicio	Fecha de entrega
2024 - B	02/09/24	03/09/24

Índice

1. Introducción

En un mundo donde la conectividad y la comunicación son esenciales para el funcionamiento de casi todas nuestras actividades cotidianas de hoy en dia, las telecomunicaciones y la conectividad con el internet se han convertido en un campo de vital importancia. La necesidad de herramientas específicas para abordar los desafíos técnicos de este sector ha impulsado al desarrollo de **NetCode** en una primera instancia, en la cual esta desarrollado en un enfoque particular y amigable a las **telecomunicaciones y al mundo WISP**. Este informe se enfoca principalmente en presentar **NetCode**, describiendo su relevancia en el contexto actual, sus objetivos y la propuesta que ofrece para mejorar la programación en este campo.

1.1. Justificación

La rápida evolución tecnológica en **telecomunicaciones** ha incrementado significativamente la complejidad de los sistemas de comunicación. Las herramientas de programación actuales frecuentemente no satisfacen plenamente las demandas específicas del sector, especialmente en áreas críticas como la manipulación de señales, la gestión de paquetes de datos y la simulación de redes.

Además, **los proveedores de servicios de Internet inalámbrico (WISP)** enfrentan desafíos únicos. Estos proveedores deben gestionar la transmisión de datos en entornos variables, lidiar con interferencias de señal y optimizar el uso del espectro radioeléctrico. La naturaleza dinámica y a menudo impredecible de las redes inalámbricas exige soluciones especializadas que permitan una gestión eficiente y efectiva.

1.2. Objetivo

En este contexto, **NetCode** tiene como objetivo brindar una sintaxis específicamente adaptada para abordar las necesidades del sector de **telecomunicaciones** y, en particular, para el mundo **WISP**, **NetCode** ofrece una sintaxis elaborada para este ambito, que a futuro tambien pueda ofrecer herramientas de gestión, seguridad informatica, transmición de datos, optimización de redes inalámbricas y la gestión de sistemas de comunicación avanzados.

2. Propuesta

La propuesta actual de **NetCode** se basa principalmente en la integración de funcionalidades específicas para las telecomunicaciones, y se fundamenta en las características de los lenguajes **C++ y Go (Golang)**. **NetCode** aprovechará la robustez y el control de bajo nivel de **C++** para la manipulación eficiente de señales y la gestión de datos, mientras que se beneficiará de la simplicidad y las capacidades de concurrencia de **Go** para la simulación detallada de redes y la gestión de paquetes de datos.

3. Especificación lexica

En esta sección, se detallará la especificación léxica de **NetCode**, abordando los aspectos fundamentales que definen el conjunto de símbolos y las reglas de formación de los elementos del lenguaje. A continuación, se detalla cada punto:

3.1. Definición de los comentarios

La sintaxis de uno o varios comentarios en **NetCode** (realizados de manera lineal) será de la siguiente manera:

Listing 1: Comentarios en NetCode

```
// programa NetCode, hola mundo

function main [] {
   log["hola mundo"];
   echo;
}
```

Los comentarios en **NetCode** seran escritos después de poner //

3.2. Definición de los identificadores

Los identificadores (id) en **NetCode** tendran las siguientes características:

- Los identificadores deben iniciar siempre con una letra, ya sea minuscula o mayúscula.

- Después, pueden incluir o: números (0-9), letras reconocidas en el abecedario (A-Z) o (a-z) o sub guiones (_).

- No pueden existir espacios en blanco en los identificadores.

- Los identificadores **no** pueden ser palabras reservadas de **NetCode**.

- Toda sentencia, creación de un identidicador o asignación de un identificador debe terminar con ;.

Ejemplo:

Listing 2: Identificadores en NetCode

```
// programa NetCode, creacion y asignacion de ids

function main [] {
   numero_01 interger;
    Nsa1_ texto;
    a01 decimal = 10.90;
   echo;
}
```

3.3. Definición de las palabras clave

Las palabras clave en **NetCode** serán las siguientes:

- **function** : representa la definición de una función en **NetCode**.

- **main** : representa el punto de entrada principal del programa en **NetCode**.

- **void** : representa que una función no retorna ningún valor en especifico en **NetCode**.

- **log** : representa el poder imprimir un ente en **NetCode**, ya sea variables, números, funciones, etc.

- **echo** : representa (en algunos casos) el retorno de un valor especifico.

- **stop** : representa (en bucles) de **NetCode** el detenimiento del mismo.

- **for** : representa el inicio de una iteración definida en en **NetCode**.

- **if** : representa la evaluación de una condición en **NetCode**, si se llega a cumplir se ejecutará un bloque de código.

- **or** : se utiliza para realizar una operación lógica de disyunción en **NetCode**.

- **true** : representa un valor lógico de afirmación o veracidad. Indica que una condición o expresión lógica es verdadera en **NetCode**.

- **false** : representa un valor lógico de negación o falsedad. Indica que una condición o expresión lógica no es verdadera en **NetCode**.

- **not** : representa un valor lógico de negar un operador booleano en **NetCode**.

- **and** : se utiliza para realizar una operación lógica de conjunción en **NetCode**.

- **else** : se utiliza en estructuras condicionales para especificar un bloque de código que se ejecuta si la condición del **if** no se cumple.

- **elif** : se usa como una condición alternativa en una estructura **if-else** en **NetCode**.

- **while** : inicia un bucle que se ejecuta mientras se cumpla una condición en **NetCode**.

- **integer** : representa un tipo de dato para números enteros en **NetCode**.

- **decimal** : representa un tipo de dato para números de punto flotante en **NetCode**.

- **boolean** : representa un tipo de dato para valores lógicos, es decir, **true** o **false** en **NetCode**.

- **text** : representa un tipo de dato para cadenas de caracteres en **NetCode**.

Esta lista define las palabras clave del lenguaje **NetCode** y sus funciones específicas.

Ejemplo:

Listing 3: Algunas palabras clave en NetCode

```
// programa NetCode, algunas palabras clave

function factorial[n interger] interger {
   result interger = 1;
   for[i interger;i<=n;i++] {
      result = result * i;
   }
   echo result;
}

function main [] {
   numero interger = 5;
   log["El factorial de " $ numero " es: " $ factorial[numero]];
   echo;
}
```

3.4. Definición de los literales

Los literales en **NetCode** son valores constantes utilizados directamente en el código fuente y representan datos específicos:

- **Literales numéricos** : representan valores numéricos y pueden ser enteros o de punto flotante. Por ejemplo, `42` o `3.14`.

- **Literales de cadenas** : representan secuencias de caracteres y se encierran entre comillas dobles. Por ejemplo, `"Hello, World!"` y `"1234"`.

- **Literales booleanos** : representan valores lógicos, específicamente `true` o `false`.

3.5. Definición de los operadores

En **NetCode**, los operadores son símbolos que realizan operaciones sobre uno o más operandos. Estos operadores permiten la manipulación de datos y la implementación de lógica en el código. A continuación, se describen los principales operadores disponibles en **NetCode**:

3.5.1. Operadores aritméticos

Se utilizan para realizar operaciones matemáticas básicas. Tales como:

`+` : suma dos operandos. Ejemplo: `a + b`.

`-` : resta el segundo operando del primero. Ejemplo: `a - b`.

`*` : multiplica dos operandos. Ejemplo: `a * b`.

`/` : divide el primer operando por el segundo. Ejemplo: `a / b`.

`%` : devuelve el residuo de la división entre dos operandos. Ejemplo: `a% b`.

3.5.2. Operadores de comparación

Se utilizan para comparar dos operandos y retornan un valor booleano. Tales como:

`==` : verifica si dos operandos son iguales. Ejemplo: `a == b`.

`!=` : verifica si dos operandos son diferentes. Ejemplo: `a != b`.

`>` : verifica si el primer operando es mayor que el segundo. Ejemplo: `a >b`.

`<` : verifica si el primer operando es menor que el segundo. Ejemplo: `a <b`.

`>=` : verifica si el primer operando es mayor o igual que el segundo. Ejemplo: `a >= b`.

`<=` : verifica si el primer operando es menor o igual que el segundo. Ejemplo: `a <= b`.

3.5.3. Operadores lógicos

Se utilizan para combinar expresiones booleanas. Tales como:

`and` : realiza una operación lógica de conjunción. Ejemplo: `a and b`.

`or` : realiza una operación lógica de disyunción. Ejemplo: `a or b`.

`not` : realiza una operación lógica de negación. Ejemplo: `not a`.

3.5.4. Operadores de asignación

Se utilizan para asignar valores a variables. Tales como:

`=` : asigna el valor del operando derecho al operando izquierdo. Ejemplo: `a = b`.

`+=` : suma el operando derecho al operando izquierdo y asigna el resultado al operando izquierdo. Ejemplo: `a += b`.

`-=` : resta el operando derecho del operando izquierdo y asigna el resultado al operando izquierdo. Ejemplo: `a -= b`.

`*=` : multiplica el operando izquierdo por el operando derecho y asigna el resultado al operando izquierdo. Ejemplo: `a *= b`.

`/=` : divide el operando izquierdo por el operando derecho y asigna el resultado al operando izquierdo. Ejemplo: `a /= b`.

3.5.5. Operadores de incremento y decremento

Se utilizan para aumentar o disminuir el valor de una variable en una unidad. Tales como:

`++` : incrementa el valor de una variable en 1. Ejemplo: `a++`.

`--` : decrementa el valor de una variable en 1. Ejemplo: `a--`.

Ejemplos:

Listing 4: Algunos operadores en NetCode

```
function checkNumber[num interger] void {
    // Evaluacion de una condicion usando if-else
    if [num > 0] {
        // Bloque de codigo que se ejecuta si la condicion es verdadera
        log["El numero " $ num " es positivo"];
    } elif [num == 0] {
        // Bloque de codigo que se ejecuta si la condicin es falsa pero la condicin del
            else-if es verdadera
        log["El numero " $ num " es cero"];
    } else {
```

```
        // Bloque de codigo que se ejecuta si ninguna de las condiciones anteriores es
            verdadera
        log["El numero " $ num " es negativo"];
    }
}
```

Listing 5: Algunos operadores en NetCode

```
function main [] {
    // Declaracion e inicializacion de variables
    a integer = 10;
    b integer = 5;

    // Operador de asignacion simple
    c interger = a; // Asigna el valor de a a c

    // Operador de asignacion con suma
    a += 5; // Equivalente a a = a + 5; Ahora a es 15

    // Operador de asignacion con resta
    b -= 2; // Equivalente a b = b - 2; Ahora b es 3

    // Operador de asignacion con multiplicacion
    a *= 2; // Equivalente a a = a * 2; Ahora a es 30

    // Operador de asignacion con division
    b /= 1; // Equivalente a b = b / 1; b sigue siendo 3

    // Imprimir los resultados
    log["Valor de a despues de +=: " $ a];
    log["Valor de b despues de -=: " $ b];
    // Finalizacion del programa
    echo;
}
```

Listing 6: Algunos operadores en NetCode

```
function isPrime[num integer] boolean {
    // Un numero menor o igual a 1 no es primo
    if [num <= 1] { echo false;}
    // Verificar si el numero es divisible por algun numero menor que el
    for[i integer = 2; i * i <= num; i++] {
        if [num % i == 0] {
            echo false;
        }
    }
    echo true;
}
```

4. Expresiones regulares

Según lo desarrollado en clase, tenemos las siguientes expresiones regulares:

Token	Expresión regular
FUNCION	function
PRINCIPAL	main
CORCHETEABI	[
CORCHETECERR	]
IMPRIMIR	log
COMA	,
ID	[A-Z-a-z]$^+$([$A-Z$-$a-z$] | [$0-9$] | _)*
FINALSENTENCIA	;
RETORNAR	echo
DETENER	stop
LLAVEABI	{
LLAVECERR	}
TIPOENTERO	interger
TIPOCADENA	text
TIPODECIMAL	decimal
TIPOBOOLEANO	boolean
TIPOVACIO	void
SI	if
Y	and
O	or
NO	not
SINO	elif
ENTONCES	else
MIENTRAS	while
PARA	for
SUMA	+
RESTA	-
MULTIPLICACION	*
DIVISION	/
RESIDUO	%
MENORQUE	<
MAYORQUE	>
MENORIGUALQUE	<=
MAYORIGUALQUE	>=
IGUAL	=
IGUALBOOLEANO	==
DIFERENTEDE	!=
AUMENTAR	++
DISMINUIR	--
SUMAIGUAL	+=
RESTAIGUAL	-=
MULTIPLICACIONIGUAL	*=
DIVISIONIGUAL	/=
CONCATENAR	$
NENTERO	[0-9]$^+$ | $-[0-9]^+$
NDECIMAL	[0-9]$^+$.[$0-9$]$^+$ | $-[0-9]^+$.[$0-9$]$^+$
NCADENA	" (^") "
NBOOLEANO	true | $false$
COMENTARIO	// .*