Universidad Complutense

Procesadores del lenguaje

Práctica 1

Sintaxis del lenguaje CodeX

Autores:
Marta Vicente Navarro
Pablo Navarro Cebrián

13 de mayo de 2024

Índice

1. Introducción

Este documento es la entrega final para la asignatura de *Procesadores del lenguaje* por el grupo formado por Marta Vicente Navarro y Pablo Navarro Cebrián. La entrega consiste en la definición de un mini-lenguaje de programación, y el desarrollo de un compilador para el mismo. En nuestro caso, hemos llamado al lenguaje que vamos a desarrollar CodeX. Este lenguaje está basado en otros como C++, Java o Python y trata de unificar las ventajas de cada uno de ellos de la forma más sencilla posible. Este documento recoge la especificación de la sintaxis del lenguaje de programación a tratar y de ejemplos típicos de programas escritos en dicho lenguaje.

2. Cambios realizados

No ha habido muchos cambios respecto a la primera entrega. Se ha añadido la opción de crear alias. También hemos añadido poder imprimir booleanos. Por último, las funciones sólo devolveran tipos básicos.

3. Identificadores y ámbitos de definición

- Los identificadres en CodeX empezarán por una letra (tanto mayúscula como minúscula) y podrán ir seguidos de cualquier combinación de letras, dígitos y '_' sin espacios en blanco.
  Un ejemplo de identificador válido es: num_Maximo_Coches1
  Un identificador inválido podría ser: 1coche (empieza con un dígito).

- En nuestro lenguaje se podrán declarar variables de la siguiente manera:

```
tipo identificador;
```

  donde el tipo será un tipo básico predefinido o una estructura.

- También se podrán declarar arrays. Serán estáticos, por lo que al declararlos habrá que especificar el tamaño.
  Se declararán de la siguiente manera:

```
listX<tipo>[N] identificador;
```

  donde tipo podrá ser un tipo básico predefinido, una estructura o un array.

- En nuestro lenguaje se permitirán bloques anidados como, por ejemplo, un 'if' dentro de un 'while'. Todos los bloques irán entre llaves { }. Cada vez que se entra en un nuevo bloque, el ámbito de definición cambiará al bloque en el que se ha entrado. Si se declaran variables con un mismo identificador en un bloque más 'profundo', estas ocultarán a las declaradas en bloques más 'superficiales'.

- También se podrán crear funciones, cuyos parámetros se podrán pasar tanto por valor como por referencia. Los argumentos que se pasen por referencia habrá que identificarlos en la declaración de la función añadiendo un '&' tras el tipo. Las funciones tendrán la siguiente estructura:

```
tipo func(tipo [&] param1, ..., tipo [&] paramN) {
    //Lista de instrucciones
    return tipo;
}
```

- Por último, también se podrán declarar estructuras. Estas se declaran de la siguiente manera:

```
structX nombreEstructura {
    // Definicion de subvariables
};
```

4. Tipos

En CodeX, el usuario debe declarar los tipos explícitamente. A lo largo de esta sección veremos cuál es la sintáxis para hacerlo así como los distintos tipos con los que cuenta el lenguaje. Adicionalmente, se ha de tener en cuenta que en CodeX, cada instrucción va seguida del símbolo terminal ;. Además el nombre que se le pone a una variable al declararla debe estar formado por caracteres alfanuméricos y guiones bajos y además es obligatorio que comience por una letra, bien mayúscula o bien minúscula.

4.1. Tipos básicos predefinidos

A continuación se detallan los tipos básicos predefinidos en CodeX:

- intX : representa enteros
- boolX : representa los valores True o False
- voidX : representa el tipo vacío

4.2. Tipos definidos por el usuario

4.2.1. StructX

Los usuarios de CodeX pueden crear sus propias estructuras. Para definirlas se hace uso de la palabra reservada structX que define una estructura formada por campos, que son una o más variables de uno o distintos tipos. Estos campos van entre llaves y detrás de la última llave se debe incluir ";". La sintaxis es:

```
structX nombreEstructura {
    // Definicion de subvariables
};
```

Un ejemplo concreto del uso de esta estructura sería un stuctX que represente un coche y que guarde atributos como la velocidad máxima, el número de muertas o si esta en movimiento:

```
structX Coche {
    intX velocidadMaxima;
    boolX moviendose;
    intX numPuertas;
};
```

4.2.2. AliasX

Por último, nuestro lenguaje permite definir alias de tipos. La sintaxis es:

```
aliasX id = tipo;
```

Sintaxis alias

Un ejemplo concreto de alias es:

```
aliasX ent = intX;
aliasX lista10ent = listX<ent>[10];
```

Ejemplo alias

4.3. Tipo array

El tipo array representa una lista de elementos del mismo tipo en CodeX. Es importante destacar que en CodeX, los arrays no pueden contener elementos de tipos distintos. Para declarar arrays, utilizamos la palabra reservada listX. A continuación se muestra la sintáxis para definir arrays, siendo tipo el tipo de los elementos y N el tamaño del array, que debe ser un número entero. Esto permite generar arrays de diferentes dimensiones e incluso arrays de arrays de tamaño M, es decir, matrices. La sintáxis para ambos casos sería:

```
listX<tipo> nombreLista[N];
listX<listX<tipo>[N]>[M] nombreMatriz;
```

A continuación se muestra un ejemplo concreto de definición de arrays y matrices. También es posible inicializar un array utilizando los corchetes [ ].

```
listX<coche> concesionario[N];
listX<listX<coche>[N]>[M] concesionariosBMW;
listX<listX<intX>[3]>[3] identidad = [[1,0,0],[0,1,0],[0,0,1]].
```

En CodeX, los arrays son estáticos, lo que significa que su tamaño está ligado a su tipo. Por esta razón, el tamaño debe especificarse siempre, incluso si el array es un parámetro de función.

4.4. Tipo Puntero

En CodeX se pueden definir punteros a datos mediante el tipo puntero. Para declarar una variable de tipo puntero, se escribe * entre el tipo y el identificador. La sintaxis es la siguiente:

```
tipo* punteroATipo;
```

Un ejemplo concreto de ello es:

```
intX* punteroAInt;
```

También se podrá reservar memoria dinámica mediante la palabra reservada new. El resultado será un puntero a un tipo concreto. La sintaxis es la siguiente:

```
tipo* punteroATipo = new tipo;
```

Por último, el operador unario prefijo & se puede usar para obtener la dirección de memoria de una variable y el operador unario prefijo * se puede usar para obtener el contenido de una variable de tipo puntero. Ambos operadores son asociativos. Un ejemplo de su uso es:

```
intX* punteroATipo = new intX;
*punteroATipo = 3;

intX numero = 5;
intX* punteroANumero = &numero;

intX numero2 = *punteroANumero;
```

4.5. Operadores

Los operadores son símbolos o palabras clave que se utilizan para realizar operaciones sobre datos y variables. A continuación se indica una tabla resumen de los operadores de CodeX junto con su prioridad, asociatividad y tipo.

Operador	Prioridad	Tipo	Asociatividad
or	0	Binario infijo	Por la izquierda
and	1	Binario infijo	Por la izquierda
equals, not_equals	2	Binario infijo	Por la izquierda
<, >, <=, >=	3	Binario infijo	Por la izquierda
new	3	Unario prefijo	No asociativo
+, -	4	Binario infijo	Por la izquierda
*, /, %	5	Binario infijo	Por la izquierda
not, -	6	Unario prefijo	Por la derecha

Cuadro 1: Tabla de Operadores

Los tipos sobre los que pueden actuar los operadores son:

- or : boolX or boolX
- and : boolX and boolX
- equals, not_equals : (boolX op boolX) o (intX op int X)
- <, >, <=, >= : intX op int X
- new : new tipo
- +, - : intX op intX
- *, /, % : intX op intX
- not, -: not boolX, - intX

5. Conjunto de instrucciones del lenguaje

Las instrucciones en un lenguaje de programación son las órdenes que se le dan a la computadora para que realice determinadas acciones. En esta sección, se presentan las instrucciones del lenguaje CodeX.

5.1. Expresiones

Las expresiones son secuencias de operadores y operandos que se utilizan para diversos propósitos. En el caso de CodeX las expresiones pueden contener:

- Constantes: tanto de tipo entero $\in \mathbb{Z}$ como booleano $\in \{\text{True, False}\}$.
- Variables: dentro de la expresión se referencia a la variable con su identificador, previamente definido.
- Elementos de arrays: dentro de la expresión se referencia al elemento mediante un acceso al array utilizando el índice. La sintaxis de un acceso a un array es la siguiente:

```
1 array[posicion];
```

Acceso a un elemento de un array

  En los accesos a arrays, la posición debe ser un número entero entre 0 y el tamaño del array menos 1 (ambos incluidos).

- Un acceso a un campo de una estructura: se accedera de la siguiente manera:

```
1 nombreEstructura->campo;
```

Acceso a un campo de una estructura

- Llamada a una función

5.2. Declaración y asignación

Una declaración es una instrucción que genera una nueva variable. En CodeX existen dos tipos de declaraciones: con o sin valor inicial.

Cuando queremos darle un valor a una variable existente podemos realizar una asignación. Además, también podemos asignar valores a elementos de un array.

A continuación se muestra la estructura y un ejemplo de declaraciones con y sin valor inicial en CodeX y asignaciones a variables ya existentes.

```
1 tipo nombre1; // sin valor inicial
2 tipo nombre2 = valor1; // con valor incial
3 nombre1 = valor2;
4 array[posicion] = valor3;
```

Estructura de declaración y asignación

```
intX m;
intX n = 10;
m = 5;
a[0] = 7; //a es un array ya existente
```

Ejemplos de declaración y asignación

5.3. Condicional

Una instrucción condicional en CodeX es una estructura que permite ejecutar diferentes bloques de código según se cumplan ciertas condiciones. Básicamente, decide qué acción tomar en función de si una condición es verdadera o falsa. En CodeX existen dos tipos: condicionales de una rama y condicionales de dos ramas. A continuación se presenta cada uno de estos tipos.

5.3.1. Condicional de una rama

Los condicionales de una rama permiten que un bloque de instrucciones se ejecuten solo en caso de que se satisfaga una determinada condición.

```
if(condicion){
// Bloque de instrucciones
}
```

Estructura de un condicional de una rama

```
if(x < 5){
    x = x + 1;
}
```

Ejemplo de un condicional de una rama

5.3.2. Condicional de dos ramas

Los condicionales de dos ramas permiten que, en función de si se satisface una condición o no, se ejecute un bloque de código u otro.

```
if(condicion){
// Bloque de instrucciones exclusivo para cuando se satisface la condicion
}
otherwise{
//Bloque de instrucciones exclusivo para cuando no se cumple la condicion
}
```

Estructura de un condicional de dos ramas

```
if(x < 5){
    x = x + 1;
}
otherwise{
    x = x - 2;
}
```

Ejemplo de un condicional de dos ramas

5.4. Bucles

Los bucles en programación son estructuras que permiten ejecutar un bloque de código repetidamente hasta que se cumple una condición específica. A continuación se presentan los dos tipos de bucles que existen en CodeX.

5.4.1. While

Los bucles de tipo *while* ejecutan un bloque de código mientras una condición sea verdadera. Aquí se presenta su estructura y un ejemplo.

```
while(condicion){
// Lista de instrucciones si se cumple la condicion
}
```

Estructura de un bucle While

```
while(x < 5){
    x = x + 1;
}
```

Ejemplo de un bucle While

5.4.2. For

Los bucles de tipo *for* se ejecutan para todo un rango de valores que viene determinado por tres instrucciones separadas por ; y que se realizan en el siguiente orden:

- Declaración e inicialización de la variable sobre la que se itera
- Condicion: Expresión booleana que determina la condición de parada del bucle.
- Modificación: determina la modificación que sufre la variable declarada en el paso 1 tras cada iteración.

Es decir, la estructura de un bucle for quedaría de la siguiente manera:

```
for(declaracion; condicion; modificacion){
// Lista de instrucciones si se cumple la condicion
}
```

Estructura de un bucle For

```
for(intX i = 0; i < 5; i++){
    x = x + 1;
}
```

Ejemplo de un bucle For

Cabe destacar que en CodeX la declaración de la variable sobre la que itera el bucle es exclusiva del mismo de forma que no podría ser utilizada fuera de él.

5.5. Return

En CodeX, podemos clasificar las funciones en dos tipos según su finalización y valor de retorno.

En primer lugar, las funciones de tipo vacío. Aunque no devuelven nada, será obligatorio añadir la instrucción return;. Esta deberá ser única en la función y estar al final del cuerpo de la misma, fuera de cualquier bloque interno.

```
voidX nombreFuncion(argumentos){
    //Lista de instrucciones
    return;
}
```

Estructuras de función de tipo vacio

En segundo lugar, las funciones de tipo no vacío tendrán que finalizar siempre con una instrucción de tipo return tipo; siendo *tipo* el tipo de la función. Al igual que en el primer caso, la instrucción return deberá ser única en la función y estar al final del cuerpo de la misma, fuera de cualquier bloque interno.

```
tipo nombreFuncion(argumentos){
    //Lista de instrucciones
    return tipo;
}
```

Estrutura de una función de tipo no vacio

A continuación se muestran ejemplos concretos de las diferentes estructuras.

```
voidX mejorarVelocidad(Coche& c) {
    if (c->velocidadMaxima not_equals 120) {
        c->velocidadMaxima = c->velocidadMaxima + 5;
    }

    return;
}

intX consultarNumPuertas(Coche c){
    return c-> numPuertas;
}
```

Ejemplos concretos

5.6. Funciones

En este lenguaje se pueden definir funciones y realizar llamadas a funciones como parte de una expresión. Esto implica que en caso de asignarle a una variable el valor de retorno de una función, ha de ser del mismo tipo. La función principal del código es la función main que ha de estar siempre definida y es por la que comienza la ejecución. Es de tipo intX. Además, debe ser siempre la última declaración.

```
tipo funcion1(argumentos){
    //Instrucciones
}
```

Estructura de función que llama a otra

```
tipo funcion2(argumentos){
    //Instrucciones
    tipo variable = funcion1(argumentos);
    //Instrucciones
}
```

Ejemplo de función que llama a otra

```
intX main() {
    //Instrucciones
    return 0;
}
```

Sintaxis función main

5.7. Entrada y salida

En nuestro lenguaje la interacción con la consola se hace mediante las funciones readIntX y writeX ya que CodeX solo permitirá leer enteros y escribir tanto enteros como booleanos (1 si es True o 0 si es False). Para leer se le pasa como argumento la variable en la que quieres almacenar el dato leído, y para escribir se le pasa como argumento una expresión de tipo entero o bool. Es decir, estos serían ejemplos de lectura y escritura:

```
intX x;
readIntX(x);
x = x + 5;
writeX(x);
writeX(x + 24 - 3);

boolX bool = True;
writeX(bool);
writeX(bool and False);
```

Ejemplo de lectura y escritura

6. Ejemplos

En esta sección se presentan algunos ejemplos de programas típicos escritos en CodeX.

6.1. Tipos definidos por el usuario y E/S

A continuación, se muestra un ejemplo de código que crea un *coche*, lo acelera y muestra su velocidad por consola.

```
structX Coche{
    intX velocidadMaxima;
    boolX moviendose;
    intX numPuertas;
}

voidX mejorarVelocidad(Coche &c){
    if (c->velocidadMaxima not _equals 120){
        c->velocidadMaxima = c->velocidadMaxima + 5;
    }
    return;
}

voidX cambiarVMax(Coche& c){
    c->velocidadMaxima = 100;
}

intX consultarNumPuertas(Coche c){
    return c-> numPuertas;
}

intX main(){
    Coche c;
    c->numPuertas = 5;
    c->moviendose = True;
    c->velocidadMaxima = 20 * consultarNumPuertas(c);
    mejorarVelocidad(c);
    intX vMax = c->velocidadMaxima;
    writeX(vMax);
    return 0;
}
```

Ejemplo de código en Codex

6.2. Bucles y condicionales

Siendo un coche, la estructura que se define en el ejemplo anterior, el código a continuación muestra na función que lee una velocidad y busca el primer coche del array concesionario que tiene como velocidadMaxima esa velocidad.

```
intX selecCoche(listX<Coche>[10] & concesionario, intX velocidad){

    intX index = -1;

    for(intX i = 0; i < 10; i = i + 1) {
        if (concesionario[i]->velocidadMaxima equals 45){
            index = i;;
        }
    }
    return index;
}

intX main(){
```

```
    listX<Coche>[10] concesionario;
    ... //Inicializar todos los coches
    intX velocidadBuscada;
    readIntX(velocidadBuscada);
    intX cocheBuscado = selecCoche(concesionario, velocidadBuscada);
    writeX(cocheBuscado);

    return 0;
}
```

Ejemplo de código en Codex