

Universidad Nacional de Río Cuarto  
Facultad de Ciencias Exactas Físico-Químicas y Naturales  
Departamento de Computación

## Taller de Diseño de Software

(Cod. 3306)

Descripción del Lenguaje: TDS2020

2020

---

El proyecto de la materia consiste en implementar un compilador para un lenguaje imperativo simple, similar a `C` o `Pascal`, llamado TDS2020.

### Consideraciones del Léxico

El lenguaje TDS2020 es *case-sensitive*. Las **palabras reservadas** del lenguaje únicamente están formadas por minúsculas. Las palabras reservadas y los identificadores son *case-sensitive*. Por ejemplo, **bool** es una palabra reservada, pero **BOOL** es un identificador; **cont** and **Cont** son dos nombres diferentes de variables distintas.

Las palabras reservadas son:

**integer bool false true extern return void**

Dos tipos de comentarios son permitidos, aquellos que comienzan con `//` y terminan al final de la línea (únicamente pueden ser de una línea) y aquellos que están delimitados por `/*` y `*/` (pueden tener varias líneas de extensión).

Uno o más espacios pueden aparecer entre los símbolos del lenguaje. Llamamos espacios a los espacios en blanco, tabulaciones, saltos de líneas y/o comentarios.

Las palabras reservadas y los identificadores deben estar separados por un espacio o por un símbolo que no es ni una palabra reservada ni un identificador. Por ejemplo, **whiletrue** es un identificador, no son dos palabras reservadas distintas.

Los literales (valores constantes) del lenguaje son: números enteros y valores lógicos. Los literales enteros son iguales a los utilizados en `C` (por ejemplo 123). Los números enteros son de 32 bit con signo, es decir, los valores están en el rango entre  $-2147483648$  y  $2147483647$ . Los literales booleanos son **true** y **false**.

## Gramática

### Notación:

$\langle \text{simb} \rangle$	$\langle \text{simb} \rangle$ es un no-terminal.
<b>simb</b>	<b>simb</b> a es un terminal
'x'	x es un terminal
$[x]$	cero o una ocurrencia de $x$ , <i>i.e.</i> , $x$ es opcional; notar que corchetes entre comillas '[' ']' son terminales.
$x^*$	cero o más ocurrencias de $x$ .
$x^+,$	una lista de una o más ocurrencias de $x$ 's separadas por coma.
$\{ \}$	llaves son usadas para agrupar; notar que llaves entre comillas '{' '}' son terminales.
	separa alternativas.

$\langle \text{program} \rangle \rightarrow \langle \text{var\_decl} \rangle^* \langle \text{method\_decl} \rangle^*$

$\langle \text{var\_decl} \rangle \rightarrow \langle \text{type} \rangle \{ \langle \text{id} \rangle^+, ', ' \}$

$\langle \text{method\_decl} \rangle \rightarrow \{ \langle \text{type} \rangle \mid \mathbf{void} \} \langle \text{id} \rangle ' ( ' [ \langle \text{type} \rangle \langle \text{id} \rangle^+, ', ' ] ' ) ' \langle \text{block} \rangle$   
 $\mid \mathbf{extern} \{ \langle \text{type} \rangle \mid \mathbf{void} \} \langle \text{id} \rangle ' ( ' [ \langle \text{type} \rangle \langle \text{id} \rangle^+, ', ' ] ' ) ' ' ; '$

$\langle \text{block} \rangle \rightarrow ' \{ ' \langle \text{var\_decl} \rangle^* \langle \text{statement} \rangle^* ' \}$

$\langle \text{type} \rangle \rightarrow \mathbf{integer} \mid \mathbf{bool}$

$\langle \text{statement} \rangle \rightarrow \langle \text{id} \rangle ' = ' \langle \text{expr} \rangle ' ; '$   
 $\mid \mathbf{return} [ \langle \text{expr} \rangle ] ' ; '$   
 $\mid ' ; '$   
 $\mid \langle \text{block} \rangle$

$\langle \text{expr} \rangle \rightarrow \langle \text{id} \rangle$   
 $\mid \langle \text{literal} \rangle$   
 $\mid \langle \text{expr} \rangle \langle \text{bin\_op} \rangle \langle \text{expr} \rangle$   
 $\mid ' ! ' \langle \text{expr} \rangle$   
 $\mid ' ( ' \langle \text{expr} \rangle ' ) '$

$\langle \text{bin\_op} \rangle \rightarrow \langle \text{arith\_op} \rangle \mid \langle \text{rel\_op} \rangle \mid \langle \text{cond\_op} \rangle$

$\langle \text{arith\_op} \rangle \rightarrow ' + ' \mid ' * '$

$\langle \text{rel\_op} \rangle \rightarrow ' == '$

$\langle \text{cond\_op} \rangle \rightarrow ' \&\& '$

$\langle \text{literal} \rangle \rightarrow \langle \text{integer\_literal} \rangle \mid \langle \text{bool\_literal} \rangle$

$\langle \text{id} \rangle \rightarrow \langle \text{alpha} \rangle \langle \text{alpha\_num} \rangle^*$

$\langle \text{alpha\_num} \rangle \rightarrow \langle \text{alpha} \rangle \mid \langle \text{digit} \rangle \mid ' _ '$

$\langle \text{alpha} \rangle \rightarrow ' \mathbf{a} ' \mid ' \mathbf{b} ' \mid \dots \mid ' \mathbf{z} ' \mid ' \mathbf{A} ' \mid ' \mathbf{B} ' \mid \dots \mid ' \mathbf{Z} '$

$\langle \text{digit} \rangle \rightarrow ' 0 ' \mid ' 1 ' \mid ' 2 ' \mid \dots \mid ' 9 '$

$\langle \text{integer\_literal} \rangle \rightarrow \langle \text{digit} \rangle \langle \text{digit} \rangle^*$

$\langle \text{bool\_literal} \rangle \rightarrow \mathbf{true} \mid \mathbf{false}$

## Ejemplo de Programa en TDS2020

```
integer inc(integer x){
    return x + 1;
}

extern integer get_int();

extern void print_int(integer i);

integer main(){
    integer y;
    y = 1;
    {
        integer x;
        x = 2;
        y = x + 1;
    }
    return x;
}
```

## Semántica

Un programa TDS2020 consiste de una lista de declaraciones de variables globales y funciones. El programa debe contener la declaración de un método llamado **main**. Este método no tiene parametros. La ejecución de un programa TDS2020 comienza con el método **main**.

## Tipos

Los tipos básicos en TDS2020 son **integer** y **bool**.

## Reglas de Alcance y Visibilidad de los Identificadores

Las reglas de alcance y visibilidad en TDS2020 son simples. Primero, todos los identificadores deben ser definidos (textualmente) antes de ser usados. Por ejemplo, una variable debe ser declarada antes de ser usada; una función puede ser invocada únicamente por código ubicado después de su declaración (aunque esta primer versión no permite la invocación de funciones).

En un punto de un programa TDS2020 existen al menos dos ámbitos (*scopes*) válidos, el global y el local a la función. El scope global esta conformado por los identificadores de las variables y de las funciones declaradas al definir el programa. El scope de la función esta conformado por los parámetros formales y los identificadores de las variables declaradas en el cuerpo de la función. Se pueden definir *scope* locales adicionales al introducir bloques (*block*) de código. Los distintos *scopes* tienen una relación de anidamiento, tal que, el scope global contiene a el scope de las funciones y estos contienen a los scopes de los bloques (los cuales tambien pueden ser declarados de manera anidada). Este anidamiento causa que identificadores definidos en un scope pueda ocultar un identificador con el mismo nombre en scopes superiores. Se debe notar que una variable local puede ocultar tanto un identificador de una variable como de una función.

Los nombres de los identificadores son únicos en cada scope. Es decir, no se puede utilizar el mismo identificador más de una vez en cada ámbito. Por ejemplo, variables y funciones deben tener distinto nombre en el scope global.

## Locaciones en Memoria

El lenguaje TDS2020 tiene una clase de locaciones: variables (locales y globales). Cada locación en memoria tiene un tipo. Por ejemplo, las locaciones de tipo **integer** y **bool** contienen valores enteros y lógicos, respectivamente;

## Asignaciones

Solo se permiten asignaciones a variables de tipos básicos, es decir, variables de tipos **integer** y **bool**. La semántica de las asignaciones define la copia del valor. La asignación  $\langle \text{location} \rangle = \langle \text{expr} \rangle$  copia el valor resultante de evaluar la  $\langle \text{expr} \rangle$  en  $\langle \text{location} \rangle$  (copia el valor de la expresión en la variable). Una asignación es válida si  $\langle \text{location} \rangle$  y  $\langle \text{expr} \rangle$  tienen el mismo tipo.

Se permite asignar valores a los parámetros de un método, pero el efecto de estas asignaciones únicamente es visible en el scope del método. Los parámetros son pasados por valor.

## Expresiones.

Las expresiones siguen las reglas usuales de evaluación. En ausencia de otras restricciones, los operadores con la misma precedencia son evaluados de izquierda a derecha. Los paréntesis pueden ser usados para modificar la precedencia usual.

Una locación (variables) son evaluados al valor que contiene la locación en memoria.

Literales enteros se evalúan a su valor.

Los operadores aritméticos ( $\langle \text{arith\_op} \rangle$  y menos unario) y los operadores relacionales ( $\langle \text{rel\_op} \rangle$ ) tienen el significado y precedencia usual. `%` computa el resto de una división de números enteros.

Los operadores relacionales `<` y `>` son usados para comparar expresiones numéricas. El operador de igualdad (`==`, *es igual*) está definido para todos los tipos básicos. Únicamente se pueden comparar expresiones del mismo tipo.

El resultado de un operador relacional o de igualdad tienen tipo **bool**.

Los operadores lógicos `&&` y `||` deben ser evaluados usando evaluación de *corto circuito*. El segundo operador no es evaluado si el primer operador determina el valor de toda la expresión, es decir, si el resultado es **false** para `&&` o **true** para `||`.

Precedencia de operadores, de mayor precedencia a menor precedencia:

Operadores	Comentarios
-	menos unario
!	negación lógica
* / %	multiplicación, división, resto
+ -	suma, resta
< >	relacionales (menor, mayor)
==	igual
&&	conjunción (and)
	disyunción (or)

Notar que estas reglas de precedencia no esta reflejada en la gramática.

## Reglas Semánticas

Estas reglas son restricciones (semánticas) adicionales a las reglas sintácticas expresadas en la gramática. Un programa es válido si está bien formado gramaticalmente y no viola ninguna de las siguientes reglas. El compilador deberá verificar estas reglas, en caso de detectar que no se cumple alguna, deberá generar un mensaje de error que describa el error detectado. Si el compilador no detecta ninguna violación no deberá generar ningún informe.

1. Ningún identificador es declarado dos veces en un mismo bloque.
2. Ningún identificador es usado antes de ser declarado.
3. Todo programa contiene la definición de un método llamado **main**. Este método no tiene parámetros. Notar que la ejecución comienza con el método **main**.
4. Una sentencia **return** solo tiene asociada una expresión si el método retorna un valor, si el método no retorna un valor (es un método **void**) entonces la sentencia **return** no puede tener asociada ninguna expresión.
5. La expresión en una sentencia **return** debe ser igual al tipo de retorno declarado para el método.
6. Un  $\langle id \rangle$  usado como una  $\langle location \rangle$  debe estar declarado como un parámetro o como una variable local o global.
7. Los operandos de  $\langle arith\_op \rangle$ 's y  $\langle rel\_op \rangle$ 's deben ser de tipo **integer**.
8. Los operandos de  $\langle eq\_op \rangle$ 's (**==**) deben tener el mismo tipo (**integer** o **bool**).
9. Los operandos de  $\langle cond\_op \rangle$ 's y el operando de la negación (**!**) deben ser de tipo **bool**.
10. La  $\langle location \rangle$  y la  $\langle expr \rangle$  en una asignación,  $\langle location \rangle = \langle expr \rangle$ , deben tener el mismo tipo.