Pablo Hernández Aguado, Beatriz Herguedas Pinedo

PROCESADORES DE LENGUAJE
Segunda entrega de la práctica

▪ IDENTIFICADORES Y ÁMBITOS DE DEFINICIÓN

- **Comentarios**
  Se pueden realizar comentarios que ocupen una sola línea, y comentarios que ocupen varias líneas. Los comentarios de una línea empiezan con doble barra (//), y todo lo que se escriba después de dicha doble barra hasta el salto de línea será ignorado. Los comentarios de varias líneas se escribirán con almohadilla al principio (#) y almohadilla al final (#), siendo ignorado lo de en medio.

```
// Esto es un comentario de una línea

# Esto es un comentario
de varias líneas #
```

- **Declaración de variables simples**
  La declaración de variables sin inicializar se hará de la forma *tipo nombre*, y la declaración de variables con valor inicial se realizará de la manera *tipo nombre = valor*.

```
// Declaración de variable sin valor inicial
int x;

// Declaración de variable con valor inicial
int x = 2;
```

- **Declaración de arrays unidimensionales**
  La declaración de arrays sin valores iniciales se hará de la forma *tipo[tamaño] nombre*, y la declaración de arrays con valores iniciales se realizará de la manera *tipo[tamaño]{valor inicial} nombre*, quedando así todas las posiciones del array inicializadas al valor inicial dado.

```
// Declaración de un array sin valores iniciales
bool[10] array;

// Declaración de un array con valores iniciales
bool[10]{false} array;
```

- **Declaración de arrays multidimensionales**
  La declaración de un array de varias dimensiones sin valores iniciales será de la forma *tipo[tamaño1][tamaño2]... nombre* hasta el número de dimensiones que queramos. La declaración de un array de varias dimensiones con valores iniciales se realizará de la manera *tipo[tamaño1][tamaño2]...{valor inicial} nombre*, quedando así todas las posiciones inicializadas a valor inicial, que debe ser a su vez un array de una dimensión inferior inicializado.

```
// Declaración de un array multidimensional sin inicializar
bool[10][10] array;

// Declaración de un array multidimensional inicializado
bool[10]{false} vector;
bool[10][10]{vector} array;
```

- **Bloques anidados**
  Se podrán anidar el condicional *if* y los bucles *for* y *while*. Dichos bloques se definen con la palabra reservada correspondiente a cada uno y utilizando llaves para determinar su ámbito.

```
// Ejemplo de bucle while y condicional anidados
while (x != 0){
    if (y == 0) {
        x = 0;
    }
    else {
        x = x + 1;
    }
}
```

- **Funciones**
  Las funciones se definirán de la forma *tipo nombre(argumentos)*{}, debiendo estar el cuerpo de la función dentro de las llaves. Lo que devuelve dicha función será expresado de la manera *return valor*, siendo *valor* del tipo *tipo* expresado en la definición. Dicho *return* es único y finaliza la ejecución de la función.

```
// Ejemplo de función
bool es_par(int x){
    bool y;
    if (x % 2 == 0){ y = true; }
    else{ y =  false; }
    return y;
}
```

- **Procedimientos**
  Los procedimientos se definirán de la forma
  void nombre(argumentos){}, siendo *void* la palabra reservada para ellos.

```
// Ejemplo de procedimiento
void suma5(int x){
    x = x + 5;
}
```

- **TIPOS**

  - **Tipos básicos**
    Los tipos incluidos serán el tipo entero (*int*), el tipo booleano (*bool*), el tipo carácter (*char*) y el tipo real (*float*). El tipo *bool* tiene reservadas las palabras *true* y *false* para darle valor a sus variables. Para inicializar las variables de tipo *char* se debe poner entre comillas simples el valor. Las variables de tipo *float* se escribirán utilizando el apóstrofo (').

```
// Ejemplos de variables de cada uno de los tipos
int variable_entera = 3;
bool variable_booleana = true;
char variable_caracter = 'a';
float variable_real = 6'8;
```

  - **Operadores infijos**
    Existen varios tipos de operadores infijos.
    - Operadores unarios. Estos son el operador *más* (+), el operador *menos* (−) y el operador *negación* (*!*). La prioridad de los operadores es la estándar, como se muestra en la figura 1, donde el orden de las columnas es *prioridad, símbolo, descripción, asociatividad*.

2	+ - !	Signo (unario), negación lógica	Derechas

Figura 1: Tabla de prioridad de operadores unarios

```
// Ejemplo operadores unarios
int a = + 2;
int b = - 1;
bool c = false;
bool d = !c;
```

- Operadores aritméticos. Estos son el operador *suma* $(+)$, el operador *resta* $(-)$, el operador *producto* $(*)$, el operador *división entera* $(/)$, el operador *división decimal* (div), y el operador *módulo* $(\%)$. La prioridad de los operadores es la estándar, como se muestra en la figura 2, donde el orden de las columnas es *prioridad, símbolo, descripción, asociatividad*. La prioridad de la división decimal es la misma que la de la división entera.

3	* / %	Producto, división, módulo (resto)	Izquierdas
4	+ -	Suma y resta	Izquierdas

Figura 2: Tabla de prioridad de operadores aritméticos

```
// Ejemplo operadores aritméticos
int a = 2 + 2;
int b = 3 - 1;
int c = 9 * 9;
int d = 8 / 4;
int e = 6 % 2;
float d = 9 div 4;
```

- Operadores relacionales. Estos son el operador *menor* $(<)$, el operador *mayor* $(>)$, el operador *menor o igual* $(<=)$, el operador *mayor o igual* $(>=)$, el operador *igual* $(==)$ y el operador *distinto* $(!=)$. La prioridad de los operadores es la estándar, como se muestra en la figura 3, donde el orden de las columnas es *prioridad, símbolo, descripción, asociatividad*.

6	< <= >= >	Comparaciones de superioridad e inferioridad	Izquierdas
7	== !=	Comparaciones de igualdad	Izquierdas

Figura 3: Tabla de prioridad de operadores relacionales

```
// Ejemplo operadores relacionales
if (a < 10){ a = a + 1;}
if (b > 10){ b = b + 1;}
if (c <= 9){ c = c + 1;}
if (d >= 9){ d = d + 1;}
if (e == 9){ e = e + 1;}
if (f != 9){ f = f + 1;}
```

- Operadores lógicos. Estos son el operador *and* (&&) y el operador *or* (||). La prioridad de los operadores es la que se muestra en la figura 4, donde el orden de las columnas es *prioridad, símbolo, descripción, asociatividad*.

11	&&	Y (*And*) lógico	Izquierdas
12	||	O (*Or*) lógico	Izquierdas

Figura 4: Tabla de prioridad de operadores lógicos

```
// Ejemplo operadores lógicos
bool a = b && c;
bool d = e || f;
```

- **Equivalencia estructural de tipos**
  Los tipos *int* y *float* son tipos aritméticos, por lo que se pueden realizar operaciones aritméticas con ellos. El resultado estructural en ese caso es un tipo *float*.

```
// Ejemplo de suma de un tipo float con un tipo int
float x = 2 + 8'3;
```

- **Tipos con nombre y definición de tipos de usuario**
  El único tipo admitido es el tipo *struct*, que se declara de la forma *struct nombre*{};. Dentro de él se podrán declarar varios campos. Para acceder a sus campos se utilizará la sintaxix *nombre.campo*.

```
// Ejemplo del tipo struct
struct producto{
int identificador;
bool quedanExistencias;
};

// Acceso a un struct
producto libro;
int aux = libro.identificador;
```

CONJUNTO DE INSTRUCCIONES DEL LENGUAJE

- **Instrucción de asignación para variables**
  La instrucción de asignación para variables se realiza de la manera *nombre = valor* una vez la variable ya está inicializada. Existe también una asignación especial, la asignación del tipo $i = i + 1$, que podrá ser sustituida por la sintaxis $i{+}{+}$, y de forma análoga para la resta.

```
// Declaración de la variable
int x;

// Asignación de valor a la variable
x = 7;

// Asignaciones especiales
int a = 10;
a++:
int b = 2;
b--;
```

- **Instrucción de asignación para arrays**
  La instrucción de asignación para arrays unidimensionales se hará de la forma *nombre*[*i*] = *valor*, siendo *i* la posición del array a la que se quiere realizar la asignación. Para arrays multidimensionales se realizará de la manera *nombre*[*i*][*j*]... = *valor*.

```
// Declaración del array unidimensional
int[10] array;

// Asignación de valor a la posición elegida
array[3] = 6;

// Declaración del array multidimensional
int[10][10] array;

// Asignación de valor a la posición elegida
array[3][4] = 6;
```

- **Condicional con una y dos ramas**
  El condicional se definirá de la forma *if(condición)*{} *else*{}.

```
// Ejemplo de condicional
if(x == 0){
    y = y + 1;
}
else {
    y = y - 1;
}
```

- **Bucle while**
  La sintaxis del bucle while será de la forma *while(condición)*{}. El ámbito del bucle quedará definido por las llaves.

```
// Ejemplo bucle while
while(x > 0){
    x = x - 1;
}
```

- **Bucle for**
  La sintaxis del bucle for será de la forma
  for(inicialización; condición de parada; paso){}.
  El cuerpo del bucle se encontrará dentro de las llaves.

```
// Ejemplo bucle for
int x = 0;
for (int i = 0; i < 10; i++){
    x = x + i;
}
```

- **Expresiones formadas por constantes**
  Las constantes pueden formar parte de una expresión.

```
// Ejemplo de constante dentro de una expresión
int x = x + 2;
```

- **Identificadores**
  Los identificadores deberán tener como primer carácter una letra, pudiendo después añadir más letras, guiones bajos o números. Las letras pueden ser minúsculas o mayúsculas.

```
// Ejemplo de identificadores
int multiplo3;
bool es_par;
float pesoEnKilos;
```

- **Switch**
  El *switch* se definirá de la forma
  switch(x){*case 'a':..break; case 'b':..break;..default:..* }, donde a y b son los posibles valores que puede tomar la variable x, y siendo el caso por defecto *default* obligado.

```
// Ejemplo de switch
char x;
switch (x)
{
 case 'a':
    y = y + 1;
    break;
 case 'b':
    y = y - 1;
    break;
 default:
    y = 0;
}
```

- **Llamadas a procedimientos y funciones**
  Las llamadas a procedimientos y funciones se realizarán de la forma *nombre(argumentos)*.

```
// Ejemplo de llamada a una función con un argumento
if (es_par(x)){
    x = x + 1;
}

// Ejemplo de llamada a una función con tres argumentos
int x = ec_segundo_grado(3, 5, 1'9);
```

- **GESTIÓN DE ERRORES**
  Se indicará el tipo de error, la posición (fila y columna), y se detendrá la compilación.

- **EJEMPLO COMPLETO**
  El siguiente es un ejemplo de un programa que podría ser compilado por nuestro compilador. Dicho ejemplo trata de saber si una serie de números mantienen un orden estricto entre ellos, ya sea ascendente o descendente.

```
int main() {
        int[10] vector;
        bool ok;
        for (int i = 0; i < 10; i = i + 1) { vector[i] = i; }
        if (vector[0] > vector[1]) {
                int i = 0;
                ok = true;
                while (i < vector[10] - 1 && ok) {
                        if (vector[i] <= vector[i + 1]) {
                                ok = false;
                        }
                        i = i + 1;
                }
        }
        else {
                if (vector[0] < vector[1]) {
                        int i = 0;
                        ok = true;
                        while (i < vector[10] - 1 && ok) {
                                if (vector[i] >= vector[i + 1]) {
                                        ok = false;
                                }
                                i = i + 1;
                        }
                }
                else { ok = false; }
        }
        return 0;
}
```
