ENTREGA 2

PABLO SANZ CAPEROTE Y DANIEL VALVERDE MENASALVAS

Queremos realizar un lenguaje imperativo inspirado en C++. Sus características principales serán las siguientes:

- No será necesario indicar un punto de entrada (main) a los programas. Las instrucciones se ejecutarán en el orden que estén escritas a excepción de las funciones.
- Solo se permitirán comentarios en linea y serán indicados con el símbolo #.
- No soportará includes, los programas se escribirán en un único fichero fuente.

SINTAXIS DEL LENGUAJE

TIPOS

Para declarar una variable de un cierto tipo es necesario escribir antes del nombre de la variable la palabra reservada que define a cada tipo. Es posible declarar variables con valor inicial (mediante el uso del carácter "=" detrás del nombre). Si no se asigna un valor se asigna uno por defecto dependiendo del tipo de la variable. Nuestro lenguaje cuenta con los siguientes tipos:

VARIABLES SIMPLES:

- **Enteros:** Se identifican con la palabra reservada *int*. El valor por defecto es 0.
- **Booleanos:** Se identifican con la palabra reservada *bool*. Únicamente existirán los valores *true* y *false* que son palabras reservadas. El valor por defecto es *false*.

```
int n = 5;
int n;
boolean b = true;
boolean x;
```

ARRAYS

Los **arrays** seguirán una lógica similar a las variables simples.

- Se pueden crear arrays de todos los tipos soportados por el lenguaje.
- Se escriben poniendo la palabra reservada del tipo de la variable seguida por su su tamaño entre corchetes y posteriormente el identificador. La longitud del mismo será el número indicado entre corchetes.
- La creación de arrays multidimensionales se hará escribiendo secuencias de corchetes con la tamaño de cada dimensión.

A su vez, el array puede ser inicializado o no. Para hacerlo, hay que añadir tras el caracter "=" los valores que tomará cada posición del array entre corchetes. En caso de que no sea inicializado, cada posición del array será inicializada por defecto.

```
int[3] n = {1, 4, 5};
int[4] n;
boolean[2][3] b;
Boolean[2][2] = [[true,false],[true,true]];
```

En nuestro compilador, todas las variables son inicialmente variables de tipo array, pero al no leer los "[ ]", el compilador entiende que no es finalmente un array y pasa a ser una variable de tipo simple.

REGISTROS

En el caso de los **structs**, se declararán entre llaves, y tras el identificador del struct, todas las variables que formen parte del struct, que pueden ser de cualquiera de los demás tipos permitidos, incluidos otros registros.

No pueden existir struct vacíos, es decir; sin ninguna variable en su interior. Y tampoco pueden existir variables constantes en su interior.

```
struct myStruct{
    int n = 7;
    boolean b;
    int[2] n;
};
```

Para declarar un struct de un tipo previamente creado se hace de la siguiente forma:

```
struct myStruct nombreStruc;
```

Para acceder a los campos del struct se utiliza el operador "**.**". Y hay que poner delante la palabra reservada "struct".

```
struc nombreStruct.n = 10;
struc nombreStruct.b = true;
```

PUNTEROS

Los **punteros** se declararán mediante el símbolo *.

Se permite apuntadores a cualquiera de los tipos definidos previamente, exceptuando punteros a otros punteros.

Podrán ser inicializados mediante el símbolo "=" y el operador *new*. Esa palabra reservada *new* se encarga de reservar la memoria necesaria para almacenar el puntero. Estará seguido del tipo del puntero y de la dimensión deseada entre corchetes. Si solo queremos un dato, podremos dejar los corchetes vacíos.

```
int* p;
int* p = new int[];
int* p = new int[7];
```

A día de hoy los hemos eliminado de nuestro procesador.

EXPRESIONES

OPERADORES

Existirán operadores que nos permitirán operar con los tipos *int* y *bool*.

Booleanos: "&", "|", "!", "==".

Enteros: "+", "-", "*", "%", "/", "<", ">", "<=", ">=", "==".

Estos operandos tienen la precedencia normal y se pueden combinar entre ellos para formar expresiones más complejas.

```
6 + 5 * 4;
true & false;
```

LLAMADAS A FUNCIÓN

Las llamadas a funciones (que devuelven un tipo) son reconocidas en nuestro lenguaje como expresiones. Se pueden combinar también con los operandos para formar expresiones más complejas.

```
4 * abs(-5);
false == esContinua(f);
```

INSTRUCCIONES

El lenguaje termina de estar compuesto por instrucciones que utilizan los tipos y las expresiones vistas anteriormente.

ASIGNACIÓN DE VARIABLES

Es el proceso de dar un valor a una cierta variable. Se escribe el nombre de la variable, seguido por un signo "=" y posteriormente el valor. Se puede ver en los ejemplos de la sección *Tipo.*

DECLARACIÓN DE VARIABLES

Las variables se declaran siguiendo los pasos indicados en el apartado *Tipos.*
Se escribe primero el tipo de la variable y seguidamente el identificador o nombre de la misma. Esta instrucción puede estar seguida de una asignación posterior o no. Si hay una asignación, se da un valor por defecto (mirar sección *Tipos).*

DECLARACIÓN DE FUNCIONES

Las **funciones** se declaran indicando el tipo que devuelven (void si no devuelven nada).

A continuación, se declaran los parámetros que recibirán con sus respectivos tipos separados por comas. Los parámetros siempre se pasarán por copia.

Las instrucciones que componen la función estarán encerradas en llaves. Si la función no es declarada como void, la última instrucción de esta deberá ser un *return* que devuelva una variable del tipo indicado previamente.

Las funciones pueden recibir y devolver parámetros de cualquier tipo existentes en el lenguaje.

```
int resta(int x, int b){
    return a+b;
}
void cuadrado (int x, int y){
    int z = x*y;
}
```

BUCLES

Se incluyen el bucle **while** que tendrá la siguiente sintaxis:

```
while (condición) {
    cuerpo
}
```

INSTRUCCIONES CONDICIONALES

El caso básico es la sentencia **if**(condición), pero también puede estar formado por **if**(condición) - **else** que es opcional**.**

```
if (condición){
    cuerpo_if
}

if (condición){
    cuerpo_if
}
else {
    cuerpo_else
}
```

Por otra parte, también existen instrucciones de **Switch.** Existe una expresión que se evalúa y varios casos con expresiones del mismo tipo que la expresión del switch. Se va comparando el valor de la expresión principal con la de casa caso y se ejecutan las instrucciones del cuerpo de ese caso. Existe una última expresión que se ejecuta si ninguna de las anteriores lo ha hecho. Su esquema general es el siguiente:

```
switch (expresion){
    case valorA {
            cuerpo_caseA
    }
    case valorB {
            cuerpo_caseB
    }
    .
    .
    .
    default {
            cuerpo_default
    }
```

```
}
```

Bloques anidados

Usaremos llaves para distinguir los bloques anidados.

La indentación no será obligatoria, pero sí recomendable.

Usaremos un ejemplo con sintaxis de if y while:

```
while (i < 4) {
    if (i > 2){
    }
}
```

Instrucción de Escritura

Esta instrucción permite escribir por pantalla la variable deseada mediante la palabra reservada "**print**(variable)". Se permite ejecutar dicha instrucción únicamente con expresiones y tipos básicos. Es decir, no se puede imprimir por pantalla directamente un array, habría que hacerlo valor por valor; o en el caso de los estruct, campo a campo.

```
print(4+3);
print(x);
```

Llamadas a procedimiento

Las llamadas a procedimientos (funciones que no devuelven ningún valor) son entendidas por el compilador como instrucciones. Se escribe el nombre de la función, seguido de toda la lista de argumentos entre paréntesis.

```
procedimiento(vector, pos[2], true, 7);
```

Ejemplos

Los ejemplos se encuentran en la carpeta test. Se ejecutan los archivos de uno en uno pasando al programa el nombre del fichero y su ruta por parámetro.