Javier Mulero Martín y Ángela Ruiz Ribera

Resumen
Especificación de la sintaxis del lenguaje `C-` y de ejemplos típicos de programas escritos en dicho lenguaje.

Índice

1. Introducción

En esta entrega vamos a especificar la sintaxis de nuestro lenguaje, C-, así como algunos ejemplos típicos de programas escritos en él. Para desarrollarlo, nos hemos basado en C++.

2. Estructura de un programa

Como nos hemos basado en C++, nuestros programas tendrán una lista de declaraciones de tipos enumerados, de tipos struct y de funciones, y necesariamente tendrá una función principal llamada main.

Los comentarios en C- comienzan con el operador //, terminando el comentario con el salto de línea.

- **Enumerados**:

```
enum NombreEnumerado = {Nombre1, Nombre2,..., NombreN};
```

- **Structs**:

```
struct nombreReg {
    // lista de declaraciones [5.1]
};
```

- **Funciones**:

```
tipoRetorno nombreFun(tipo1 arg1, ..., tipoN argN) {
    // lista de instrucciones [5]
    return valorRetorno;
}
```

- **Función `main`:**

```
void main() {
    // lista de instrucciones [5]
    return;
}
```

3. Tipos

Hay declaración explicita de tipo. Estos son los tipos que hay:

3.1. Tipos básicos predefinidos:

- `int`: para representar enteros.
- `float`: para representar números decimales.
- `bool`: para representar `true` o `false`.
- `char`: para representar caracteres.
- `void`: para representar el tipo vacío.
- `string`: para representar cadena de caracteres.

3.2. Tipos definidos por el usuario:

- `struct`: define una estructura de datos formada por *campos*, que son una o más variables de uno o distintos tipos.

  Por ejemplo:
  ```
  struct Tiempo{
       int hora;
       int minuto;
       float segundo;
  };
  ```

- `enum`: representa un conjunto de valores constantes.

  Por ejemplo:
  `enum DiasSemana = {Lunes, Martes, Miercoles, Jueves, Viernes, Sabado, Domingo};`

3.3. Tipo array:

Representa una serie de elementos del mismo tipo. Se identifica por `tipo<>` o `tipo<tamaño>`. Si el array es multidimensional y accedemos a un índice, devuelve un array de una dimensión menos, por ejemplo:

```
int<3><3><2> multiarray;
int<2> una_menos = multiarray[3][3];
// devuelve un array de dimensión 1 con dos posiciones.
```

Pueden ser multidimensionales: `tipo<tamaño1><tamaño2>...<tamañoN>`.

3.4. Tipo puntero:

representa un valor que apunta a otro valor almacenado, y se identifica por `tipo *`.

3.5. Equivalencia y comprobación de tipos

- **Equivalencia estructural de tipos:** A completar cuando lo hagamos
- **Comprobación de tipos:** A completar cuando lo hagamos

4. Expresiones y operadores

Las expresiones en nuestro lenguaje pueden ser:

- Un **identificador**: formado por letras, dígitos y barrabajas. Representan a una variable ya declarada con anterioridad. Tienen que comenzar por una letra, por ejemplo: `variable_deEjeMpLo`.
- Una **constante**:
  - de tipo `int`: dígitos, por ejemplo: `20`.
  - de tipo `float`: dígitos separados por punto (.), por ejemplo: `20.03`.

Operador	Tipo	Prioridad	Asociatividad
`^`	Binario infijo	0	De izquierda a derecha
`-, !`	Unario prefijo	1	Asociativo
`*, /, %`	Binario infijo	2	De izquierda a derecha
`+, -`	Binario infijo	3	De izquierda a derecha
`<, >, <=, >=`	Binario infijo	4	De izquierda a derecha
`==, !=`	Binario infijo	5	De izquierda a derecha
`&&`	Binario infijo	6	De izquierda a derecha
`||`	Binario infijo	7	De izquierda a derecha

Cuadro 1: Operadores de `C-`. 0 indica la máxima prioridad.

  - de tipo `bool`: `true` y `false`.
  - de tipo `char`: caracteres entre comillas simples (`' '`), por ejemplo `'a'`.
  - de tipo `string`: cadena de caracteres entre comillas dobles (`" "`), por ejemplo `"Adiós mundo"`.

- Una **llamada a una función**: se escribe el nombre de la función seguido de los parámetros entre paréntesis, por ejemplo: `devuelveDia(20, variable_deEjeMpLo)`.
- Un `new tipo()`: para guardar memoria (dinámica) al iniciar los punteros, por ejemplo: si `variable_deEjeMpLo` es de tipo `int*`, `variable_deEjeMpLo = new int();`.

5. Instrucciones

A continuación mostramos las instrucciones de este nuestro lenguaje.

Observación 1. ***(Bloques anidados)*** *Anidamos instrucciones con* `{ ... }` *, y las variables que se declaren en ese bloque sólo serán efectivas en ese bloque.*

5.1. Declaración

Podemos declarar variables de dos formas:

- No inicializadas: `Tipo nombre;`
  - Ejemplo: `int<7> vector;`
- Inicializadas: `Tipo nombre = expresión;`
  - Ejemplo: `char caracter = 'a';`

5.2. Asignación

Podemos asignar a las variables una expresión: `acceso = expresion;`
Los accesos a cada uno de los tipos se hacen de la siguiente manera:

- Variables: con el `nombre` de la variable
- Punteros: `*nombre`.
- Arrays: `nombre[indice1][indice2]..[indiceN]`
- Structs: `nombre.campo`

  Por ejemplo:
  ```
  Tiempo contador;
  contador.hora = 12; contador.minuto = 15; contador.segundo = 59.9;
  ```

Observación 2. *El acceso más prioritario es el acceso a punteros, y es el único que permite asociación por paréntesis. Ejemplo:*

```
*figura.circulo = 3;
*(figura.circulo) = 3;
```

El primer ejemplo es equivalente a ~~`(*figura).circulo = 3;`~~ *pero no permitimos esta notación.*

5.3. If

Puede ser de una rama:

```
if(expresion_bool) {
    // lista de instrucciones
}
```

o de dos ramas:

```
if(expresion_bool) {
    // lista de instrucciones
} else {
    // lista de instrucciones
}
```

5.4. While

```
while(expresion_bool) {
    // lista de instrucciones
}
```

5.5. For

La declaración del `for` solo puede ser de tipo `int` y esa variable será local al bucle.

```
for(int nombre = exp; exp_bool; asig) {
    // lista de instrucciones
}
```

5.6. Switch

```
switch(nombreVarAComparar){
    case valor1: lista de instrucciones; break;
    case valor2: lista de instrucciones;break;
    default: lista de instrucciones;
}
```

5.7. Llamada a función

Como antes, también se puede llamar a una función utilizándose como instrucción en lugar de expresión.

```
nombreFun(param1, ... paramN);
```

Los valores se pasan por valor, no por referencia.

5.8. Print

Para mostrar por consola una expresión:

```
print(expresión);
```

5.9. Return

Para salir de una función devolviendo un valor o ningún valor en las funciones `void`:

```
return expresión;
```

```
return;
```

Puede haber varios `return` en una misma función.

6. Gestión de errores

Indicamos el tipo de error, fila y columna, y se para la compilación. También hay recuperación de errores (tratar de proseguir la compilación tras un error, a fin de detectar más errores).

7. Ejemplos de programas en C-

7.1. Cálculo del máximo y mínimo de un array

```
struct Solucion { // definición del struct
        int max; // declaración de variables
        int min;
};

Solucion calcular(int vector<>, int tam) { //declaro función
        Solucion sol;
```

```
        sol.max = 0; // Accedo a sus campos y los inicializo
        sol.min = vector[0]; // accedo a la primera posicion del vector
                             // e inicializo el campo min con ese valor

        int i = 0;
        while (i < tam) { // Instrucciones while, if y anidamiento de bloques
                if (vector[i] > sol.max) {
                        sol.max=vector[i];
                }
                if (vector[i] < sol.min) {
                        sol.min = vector[i];
                }
                i = i + 1;
        }
        return sol; // valor retorno
}


void main() {
    // declaración e inicialización de un array de enteros con 7 elementos
        int<7> vector; // [ 1,2,7,6,3,4,5 ];
        int i = 0;
        while ( i < 7 ){
            vector[i] = i+1;
            i = i + 1;
        }

        Solucion sol; // declaro un struct
    // llamada a función y asigno su valor de retorno a sol
        sol = calcular(vector,7);
        print(sol.min + ' ' + sol.max);
        return; // valor retorno de la función
}
```

7.2. Tipo enumerado de figuras

```
enum TipoFigura = { Cuadrado, Triangulo, Circulo };

void main() {
        TipoFigura figura; // Definimos Figura de tipo enumerado TipoFigura
        figura = Circulo;
        switch (figura) {
            case Cuadrado: print("Cuadrado :("); break;
            case Triangulo: print("Triángulo :("); break;
            case Circulo: print("Círculo :)"); break;
            default: print("No implementado >:(");
        }
        return;
```

```
}
```