C+−

Juan Diego Barrado Daganzo, Javier Saras González y Daniel González Arbelo
4º de Carrera

27 de febrero de 2024*

Índice

1. Especificaciones técnicas del lenguaje

1.1. Identificadores y ámbitos de definición

El lenguaje posee las siguientes características:

- **Declaración de variables**: se pueden declarar variables sencillas de los tipos definidos y variables *array* de estos tipos, de cualquier dimensión. Los nombres de los identificadores han de ser expresiones alfanuméricas que no comiencen por números y que posiblemente tengan el caracter "_".

*Este documento se actualiza, para consultar las últimas versiones entrar en el enlace https://github.com/JuanDiegoBarrado/PracticaPL

- **Bloques anidados**: se permiten las anidaciones en condicionales, bucles, funciones, etc. Si dos variables tienen el mismo nombre, la más profunda (en la anidación) tapa a la más externa.
- **Funciones**: se permite la creación de funciones y la declaración implícita dentro de otras. El paso por valor y por referencia de cualquier tipo a las funciones está garantizado.
- **Punteros**: para cada tipo se puede declarar un puntero a una variable de ese tipo, mediante la asignación de su dirección de memoria a la variable puntero.
- **Registros y clases**: se incluyen dos tipos adicionales: los registros como "saco de datos" —sin métodos— y las clases, tanto con datos como con métodos de función.
- **Declaración de constantes**: se incluye la posibilidad de declarar constantes por parte del usuario.

1.2. Tipos

La declaración de tipos ha de hacerse de manera explícita y de forma previa al lugar donde se emplee el identificador, es decir, que para poder usar una variable tengo que haberla declarado antes.

1.2.1. Enteros y booleanos

Los tipos básicos del lenguaje son los enteros y los booleanos. La sintaxis de declaración de estos tipos es la siguiente:

- **Enteros**: `int var;`
- **Booleanos**: `bul var;`

Entre las operaciones habilitadas para el tipo:

$$FALTAESTO$$

1.2.2. Clases y registros

Como tipos adicionales hemos incluido los registros, las clases y las funciones. La sintaxis de declaración es la siguiente:

- **Clases**: `clas var {...};`
- **Registros**: `estrut var {...};`

Entre las operaciones habilitadas para el tipo:

$$FALTAESTO$$

1.2.3. Arrays

Todos los tipos pueden formar un array multidimensional, la sintaxis de declaración es la siguiente:

- **Array**: `Tipo[DIMENSION] var;`

Entre las operaciones habilitadas para el tipo:

$$FALTAESTO$$

1.2.4. Funciones

Las funciones se han declarado también como un tipo para poder hacer expresiones lambda y pasar funciones como argumento. La sintaxis de declaración de una función es la siguiente:

- **Funciones**: `func var(Tipo arg1, Tipo arg2, ...) : TipoRetorno {...};`

El paso de parámetros por defecto es por valor, pero puede cambiarse a por referencia añadiendo el caracter "&" al final del tipo del argumento.

1.2.5. Punteros

Pueden declarase punteros a cualquiera de los tipos definidos. La sintaxis de declaración es:

- **Puntero**[1]: `Tipo~ var`

1.3. Tipos definidos por el usuario y constantes

Adicionalmente, permitimos la definición de tipos por parte del usuario a través de la palabra reservada:

- **Definición de tipos de usuario**: `taipdef nombre expresion`.

Por último, la declaración de constantes es posible gracias a la instrucción:

- **Declaración de constantes**: `difain NOMBRE valor`

1.4. Instrucciones del lenguaje

El lenguaje tiene el siguiente repetorio de instrucciones:

- **Instrucción de asignación**: `:=`

```
int var := 3
```

- **Instrucciones condicionales**: `if-else`, `switch`

```
if (var > 3) {
    ...
}
else {
    ...
}
```

- **Instrucción de bucle**: `while`

[1] Como nota especial, la declaración de un puntero a estructuras de tipo array, se haría como `Tipo[DIMENSION] var;`.

```
guail (var > 0) {
    ...
}
```

Se incluyen además las instrucciones `breic` y `continiu`.

- **Acceso a punteros**: `~punt`

1.5. Sucio

- ~~Las constantes se declaran como los `# define` de C++, pero con `# difain`.~~
- ~~Los nombres de variable siguen los mismos patrones que en C++.~~
- ~~Los arrays como las variables normales, pero con corchetes indicando la dimensión (como en C++).~~
- ~~Indicamos que una variable es puntero con `int~ var`.~~
- ~~Paso por valor y por referencia igual que en C++.~~
- ~~Los bloques tienen como separadores las llaves.~~
- ~~Los struct se escriben `estrut`~~.
- ~~El identificador de tipo es `clas`~~.
- ~~El identificador de los tipo entero es `int` y el de los booleanos es `bul`~~.
- Operadores infijos:
  - Aritméticas como C++, añadimos el operador exponencial.
  - Asignación es `:=`.
  - Igualdad es `=` y la desigualdad como en C++.
  - Los operadores booleanos son: `an`, `or`, `sor`, `not` y menor mayor es como en C++.
  - Operador parentesis y corchetes también.
- ~~Para definir los tipos `taipdef`.~~
- ~~Para definir las funciones `func nombre(args): tipoRetorno {...}`.~~
- Las funciones de entrada salida son `cein` y `ceaut` con las funciones de modificación de entrada salida que vayamos necesitando. La intrucción de retorno para las funciones es `return valor`
- ~~Acceso a punteros `~puntero`~~
- ~~Instrucciones del lenguaje:~~
  - ~~Asignación `:=`~~
  - ~~Condicionales `if () {...} else {...}` y `suich() { queis()...}`~~
  - ~~Bucle indefinido `guail`~~
  - Bucles definido `for`.
  - ~~Sentencias de control de bucles: `breic` y `continiu`.~~

A. Ejemplos de programas habituales