# MANUAL DE CIL

## RESTRICCIONES

- Los tipos son identificadores que comienzan con mayúscula
- Las variables son todas minúsculas

## INSTRUCCIONES

### Tipos

Sección destinada a la defínción de los tipos.

Se inicializa la sección de tipos con :

```
.TYPES
```

y se agrega una nueva definición de tipo según sea necesario de la siguiente manera :

```
.TYPES

type <name> {
    # PARENT (ONLY ONE)
    parent: <None or other type>

    # ATTRIBUTES
    attribute <attribute>
    ...
    ...
    ...

    # METHODS
    method <method_name>: <static_cil_function>
    ...
    ...
    ...
}
```

Aqui vemos el ejemplo del tipo Main:

```
.TYPES

type Main {
    parent: IO

    attribute a
    attribute b

    method __init: __init_Main_type
    method __init_a_at_Main: __init_a_at_Main
    method __init_b_at_Main: __init_b_at_Main
    method abort: function_abort_at_Object
```

```
    method type_name: function_type_name_at_Object
    method copy: function_copy_at_Object
    method out_string: function_out_string_at_IO
    method out_int: function_out_int_at_IO
    method in_string: function_in_string_at_IO
    method in_int: function_in_int_at_IO
    method main: function_main_at_Main
}
```

## Datos

Sección destinada a la definición de los datos estáticos del programa.

Se inicializa la sección de datos con :

```
.DATA
```

y se agrega una nueva definición de dato según sea necesario de la siguiente manera :

```
.DATA

   # CADENAS CONSTANTES
   <constant_name> = <value>
    ...
    ...
    ...
```

Aqui vemos el ejemplo de la cadena constante "Hola Mundo":

```
.DATA

string_hola_mundo = "Hola Mundo"
```

## Código

Sección destinada a la definición de las funciones del programa. Tiene que existir una función llamada main la cual es el punto de entrada al programa.

Se inicializa la sección de código con :

```
.CODE
```

y se agrega una nueva definición de código según sea necesario de la siguiente manera :

```
.CODE

    # FUNCIONES
    function <function_name> {
        # PARAMETERS
        PARAM <parameter_name>
        ...
        ...
        ...
```

```
        # LOCAL VARIABLES
        LOCAL <local_variable_name>
        ...
        ...
        ...

        # CODE
        <instruction>
        ...
        ...
        ...

        # RETURN
        RETURN <value>
    }
```

Aquí vemos el ejemplo de la función main:

```
.CODE
function function_main_at_Main {
 PARAM self

 LOCAL local_main_at_Main_internal_0
 LOCAL local_main_at_Main_internal_1

 local_main_at_Main_internal_0 = LOAD data_0
 ARG self
 ARG local_main_at_Main_internal_0
 local_main_at_Main_internal_1 = CALL function_out_string_at_IO

 RETURN local_main_at_Main_internal_1
}
```

## KeyWords

- ABORT

  > Indica que se debe abortar la ejecución del programa

  ```
  # Ejemplo
  ABORT
  ```

- ALLOCATE

  > Reserva memoria para un objeto de tipo b y devuelve la dirección en a

  ```
  # Ejemplo
  a = ALLOCATE b
  ```

- ARG

  > Indica que el próximo llamado a **CALL** acepta el argumento siguiente al comando

```
# Ejemplo
ARG self
CALL __init_Main_type
```

- ARRAY

  > Crea un arreglo de objetos de tipo b de tamaño length devolviendo la dirección de este en a

```
# Ejemplo
a = ARRAY b length
```

- CALL

  > Indica que la siguiente expresion es una llamada a una función

```
# Ejemplo
ARG self
CALL __init_Main_type
```

- CONCAT

  > Devuelve la concatenación de los strings b y c en a

```
# Ejemplo
a = CONCAT b c
```

- COPY

  > Copia superficialmente el objeto b en a

```
# Ejemplo
a = COPY b
```

- OBJEQUAL

  > Indica si los objetos a y b son iguales guardando el resultado en c

```
# Ejemplo
c = OBJEQUAL a b
```

- EQUAL

  > Indica si a y b son iguales guardando el resultado en c

```
# Ejemplo
c = EQUAL a b
```

- FATHER

  > Devuelve el padre del tipo del siguente objeto a

```
# Ejemplo
father = FATHER a
```

- GETATTR

> Obtiene el valor del atributo c del objeto b en a

```
# Ejemplo
a = GETATTR b c
```

- GETINDEX

> Obtiene el valor objeto del índice c del arreglo b en a

```
# Ejemplo
a = GETINDEX b c
```

- GOTO

> Salto incondicional a label

```
# Ejemplo
GOTO label
```

- IFGOTO

> Si a es verdadero realiza un salto a label

```
# Ejemplo
IF a GOTO label
```

- LABEL

> Define una etiqueta en el programa

```
# Ejemplo
LABEL label_0
```

> Etiquetas son utilizadas para definir un punto de entrada a una parte del código tanto para ciclos como para anotaciones.

- LENGTH

> Devuelve la longitud del string b en a

```
# Ejemplo
a = LENGTH b
```

- LOAD

> Carga el dato de la sección .DATA data_0 en a

```
# Ejemplo
a = LOAD data_0
```

- LOCAL

  > Define una variable local llamada a

```
# Ejemplo
LOCAL a
```

- NOT

  > Devuelve la negación de b en a

```
# Ejemplo
a = NOT b
```

- PARAM

  > Indica que la función acepta el parámetro self, vinculando el valor pasado en los argumentos con dicho nombre

```
# Ejemplo
PARAM self
```

- PRINT

  > Imprime el string a

```
# Ejemplo
PRINT a
```

- PRINTINT

  > Imprime el entero a

```
# Ejemplo
PRINTINT a
```

- READ

  > Lee de consola un string guardándolo en a

```
# Ejemplo
a = READ
```

- READINT

  > Lee un entero de consola guardándolo en a

```
# Ejemplo
a = READINT
```

- RETURN

  > Indica que la función retorna a

  ```
  # Ejemplo
  RETURN a
  ```

- SETATTR

  > Asigna a al atributo c del objeto b

  ```
  # Ejemplo
  SETATTR b c a
  ```

- SETINDEX

  > Asigna el objeto c al índice b del arreglo a

  ```
  # Ejemplo
  SETINDEX a b c
  ```

- SUBSTRING

  > Devuelve la subcadena de de b a partir del índice c de tamaño length en a

  ```
  # Ejemplo
  a = SUBSTRING b c length
  ```

  En caso de ser inválida la operación el programa se detiene

- TYPEOF

  > Devuelve el tipo dinámico de b en a

  ```
  # Ejemplo
  a = TYPEOF b
  ```

- VCALL

  > Devuelve en a el resultado de llamar el método c en el tipo b

  ```
  # Ejemplo
  a = VCALL b c
  ```

- VOID

  > Devuelve null en a

  ```
  # Ejemplo
  a = VOID
  ```

  > Para indicar que una variable no tiene un valor.

## Operadores

- **+**

  > Suma de dos variables

  ```
  # Ejemplo
  a = b + c
  ```

- **-**

  > Resta de dos variables

  ```
  # Ejemplo
  a = b - c
  ```

- *****

  > Multiplicación de dos variables

  ```
  # Ejemplo
  a = b * c
  ```

- **/**

  > División de dos variables

  ```
  # Ejemplo
  a = b / c
  ```

- **<**

  > En a si b es menor que c

  ```
  # Ejemplo
  a = b < c
  ```

- **>**

  > En a si b es mayor que c

  ```
  # Ejemplo
  a = b > c
  ```

## Asignaciones

- **=**

  > Asignación de un variable a otra variable

  ```
  # Ejemplo
  a = b
  ```

## Funciones

- function

> Definición de una función

```
# Ejemplo
function main
{
    PARAM self
    LOCAL local_internal_0
    local_internal_0 = READINT
    PRINT local_internal_0
    local_internal_0 = local_internal_0 + 1
    RETURN local_internal_0
    }
```

> Para definir una función se utiliza la palabra reservada function y se le asigna un nombre.
>
> Seguido se define una lista de parámetros y una lista de variables locales.
>
> Finalmente se define el cuerpo de la función.