Symbols table

- Data definition of the elements in the symbols table
  - Motivation
  - Data selection for the ALFA language
- Symbols table design

Motivation I

- In the symbols table we will store the **required da**ta for
  - variables (global from the main program and local from functions)
  - function parameters
  - functions

- We will define the required data in the symbols table items. The decisions taken in this implementation stage will rely on the source language features and the facilities provided by the compiler

- Usually, an extensive and complex language will require to store more data in the symbols table than a simple or reduced language

Motivation II

- Examples
  - If the language supports **vectors,** we will need to store in the table the *maximum size* of each defined vector to control that the indexing operations are performed within the right range
  - If the compiler supports the feature to control that each **variable** will be initialized before being used, we will need to store in the symbols table this information. This way, each time a variable is used, the compiler will query the symbols table to check its initialization
  - We can check that the number of **parameters** in a function call matches the number of parameters of the function declaration. With this purpose, we will need to store in the symbols table the number of parameters each function declares
  - We will need to store in the symbols table the **type of each variable** to perform type checking, e.g., when an assignment is performed

Data selection for the ALFA language I

- Key access to the table
  - Identifier of a variable, parameter or function
- Kind of each element
  - We will identify three kinds of elements stored in the symbols table: variable, function parameter and function
  - It defines the kind of the identifier stored in the symbols table
  - We recommend to use these **macros**
    - `#define VARIABLE 1`
    - `#define PARAMETRO 2`
    - `#define FUNCION 3`
- Basic **ALFA** data types
  - ALFA provides two basic data types: `boolean` and `int`
  - We recommend to use these macros
    - `#define BOOLEAN 1`
    - `#define INT 2`

Data selection for the ALFA language II

- If the identifier corresponds to a **vector**, this information refers to the data type of the vector elements.
- If the identifier corresponds to a **function**, this information refers to the type of the value of function return.

- **Category**
  - Identifies the structure of the data associated to the identifier of a variable or a parameter (although the parameters we will use will only be of a scalar category)
  - We distinguish two categories: **scalar** and **vector**
  - We recommend to use these macros
    - `#define  ESCALAR 1`
    - `#define  VECTOR  2`

- **Size**
  - Only for vector identifiers
  - Represent the number of rows
  - Its range is from 1 to 64

Data selection for the ALFA language III

- **Number of parameters**
  - Only for elements of the function kind
  - Number of parameters in the function declaration
- **Parameter position**
  - Only for elements of the parameter kind
  - Position of this parameter in the parameter's list (starting from 0)
- **Number of local variables**
  - Only for elements of the function kind
  - Number of local variables in the function
- **Position of the local variable**
  - Only for elements of the local variable kind
  - Position of this local variable in the local variables declaration section of the function (starting from 1)

Symbols table design I

- The symbols table of a language with only **one scope** for all the variables can consists of only one structure storing the data required for all the variables

- The symbols table of a language with **nested scopes** needs to implement somehow the management of such scopes to properly enable the visibility of the variables/parameters/functions

Symbols table design II

- In a language like **ALFA** these conditions are fulfilled
  - In a program there are **"global" variables** (placed at the beginning of the module). These variables are visible in the "global" statements of the module and inside all the functions
  - The **functions** are placed right after the global variables without any kind of nesting (inside a function we cannot declare another function). Inside a function the global variables, the local variables, the parameters and any other previously declared function are visible
  - The **module statements** are placed right after the function declarations. The global variables and the functions are visible in these statements

Symbols table design III

- The symbols table of a language like ALFA with the visibility rules previously described can be implemented as:
  - Using two structures to store the identifiers. A structure to store the global variables and the functions, that we will call the ***global table*** and another structure to store the local variables and the function parameters that we will call the ***local table***. The symbols table is made by both of these tables with a set of rules to initialize the tables and to store and search the identifiers of both tables
  - In the first step, the global table is initialized and all the identifiers of the **global variables** are stored there. If the identifier to be stored in the table is already there, this will yield a semantic error.
  - When we need to process a **function declaration**, the identifier will be stored in the global table. The local table will be initialized and the identifier of the function, the identifiers of the local variables and the parameters will be stored there. This store process will yield semantic errors when a duplicated identifier appears

Symbols table design IV

- In the part of the function statements, when an identifier appears, its **search** in the symbols table will be as follows
  - Firstly, it will be searched in the local table
  - If it is *not found* in the local table, it will be searched in the global table
- This search order implements the hiding of a global variable by a local variable with the same name
- Example of a Hash table implementation in C:
  https://programmerclick.com/article/8762897910/