# Tabela de Símbolos (Linguagem P)

A tabela de símbolos é uma das estruturas de dados mais importantes em um Compilador porque ela é responsável por armazenar as informações sobre todos os identificadores do programa-fonte.

Deverá ser criada uma **tabela de símbolos para cada função definida/declarada no código.** Cada **função define um novo escopo e cada novo escopo deverá ter uma tabela de símbolos associada a ele** para armazenar as informações sobre todas as variáveis, parâmetros e chamadas dentro do escopo da função.

A cada vez que o analisador sintático identifica uma nova declaração de função, ele deve criar uma nova tabela de símbolos e adicionar essa tabela em um vetor ou hashmap de tabelas de símbolos.

A tabela de símbolos pode ser implementada com uma estrutura de dados do tipo Hashmap onde a chave é o lexema do identificador e os valores são um conjunto de atributos descritos a seguir.

Atributos da tabela de símbolos:

- **Nome (name):** nome da variável ou da função chamada (tipo String);
- **Tipo de dado (datatype)**: armazena o tipo de dado de uma variável (int, float ou char).
- **Se é uma variável ou parâmetro (is_param):** Flag para definir se é uma variável ou parâmetro.
- **Posição do parâmetro (pos_param)**: armazena -1 para as variáveis locais e 0 para o 1º parâmetro da função, 1 para o 2º parâmetro da função e assim por diante.
- **Vetor com referências para os registros de chamadas de função (call_refs):** Um vetor que irá guardar, para cada chamada de função, uma referência para o **registro de chamada** que armezena todas as informações sobre a chamada. Para cada chamada encontrada no código, será instanciado um novo registro e inserido no final do vetor de chamadas.

Obs: O tipo de retorno (**ret_type**) de uma função será armazenado em um atributo separado. Utilizar void para funções sem retorno.

A estrutura Registro de chamada (**FunctionRegister**) deve conter os seguintes campos:

- **Lexema com o nome da função chamada (name).**
- **Número de argumentos (num_args)**: Armazena o número de argumentos da chamada.
- **Argumentos (args)**: Vetor que armazena os argumentos da chamada. Para cada argumento, é armazenada uma string contendo o lexema do argumento. No caso de constantes literais ou expressões, na fase de geração de código, seriam gerados temporários para guardar os valores das constantes e expressões e os temporários seriam referenciados na tabela.

Deverá ser definida uma representação interna para os tipos disponíveis na linguagem. No código abaixo, por exemplo, é utilizado um enum para a definição dos tipos:

```
pub enum DataTypes {
    error = -1,
    int = 0,
    float = 1,
    char = 2,
    void = 3,
}
```

Obs: Não confundir os tipos acima com os tokens.

Com relação às constantes literais, destaca-se o seguinte:

- Uma constante CHAR_LITERAL mapeia para o tipo char;

- Uma constante INT_CONST mapeia para o tipo int;
- Uma constante FLOAT_CONST mapeia para o tipo float.

EXEMPLO:

Dado o código de exemplo abaixo, apresenta-se a seguir as tabelas de símbolos preenchidas para cada função declarada no código.

```
fn maior(a: int, b: int) → int {
    let m: int;
    m = a;
    if b > m {
        m = b;
    }
    return m;
}
```

Tabela de símbolos para a função **maior**:

| chave | name | datatype | is_param | pos_param | call_refs |
|---|---|---|---|---|---|
| a | a | DataTypes::int | true | 0 | NULL |
| b | b | DataTypes::int | true | 1 | NULL |
| m | m | DataTypes::int | false | -1 | NULL |

**ret_type**(maior): DataTypes::int

```
fn main() {
    let x, y: int;
    x = 10;
    y = 20;
    println(“{}”, maior(x,y));
}
```

Tabela de símbolos para a função **main**:

| chave | name | datatype | is_param | pos_param | call_refs |
|---|---|---|---|---|---|
| x | x | DataTypes::int | false | -1 | NULL |
| y | y | DataTypes::int | false | -1 | NULL |
| println | println | DataTypes::void | false | -1 | NULL |
| maior | maior | DataTypes::int | false | -1 | **Referência para FunctionRegister** da função maior |

**ret_type**(main): DataTypes::void

Estrutura **FunctionRegister** de maior:

| name | num_args | args |
|---|---|---|
| maior | 2 | [x, y] |