PROCESADORES DE LENGUAJE. Práctica 2. Sobre la Construcción de la Tabla de Símbolos

¿Qué es la tabla de símbolos?

La tabla de símbolos (TS) es una estructura de datos que nos permite almacenar la información de los símbolos (variables y funciones) que aparecen a lo largo del programa. Esta información es necesaria para realizar las comprobaciones semánticas que se realizan sobre el código fuente, como: declaración antes de uso, comprobaciones de tipos en expresiones y asignaciones, paso de parámetros a funciones (tipo y número), etc.

¿Cómo se implementa?

En esta práctica se implementará la TS como un vector de símbolos (podéis usar también una estructura de datos dinámica como una lista enlazada). Cada símbolo se puede implementar como una estructura tipo registro. Campos a considerar son:

```
#define MAX 1024
#define TIPO_VAR 0
#define TIPO_FUNCION 1
#define TIPO_INT 2
#define TIPO_REAL 3
#define TIPO_BOOL 4
typedef struct simb_tab {
      char nombre[MAXTAMNOMBRE];      //nombre del símbolo
      int tipo_simb;                  //TIPO_VAR, TIPO_FUNCION
      int tipo_dato;                  //tipo de dato de la variable o valor de retorno de la función:
                                      (TIPO_INT, TIPO_REAL, TIPO_BOOL)
      int nargs;                      //num. de argumentos para una función
      char nombre_arg[MAXNUMARG][MAXTAMNOMBRE]; //nombre de los argumentos para las funciones
      int tipo_arg[MAXNUMARG];        //tipo de los argumentos para las funciones
      int ambito;                     //ámbito de definición del símbolo
} SIMB_TAB;

SIMB_TAB tabla_simbs[MAX];
```

El ámbito se define como la parte del programa donde el símbolo tiene validez. Supondremos que el programa principal tiene ámbito cero. Todas las variables definidas en el programa principal tienen ámbito cero. Las variables definidas dentro de funciones (o procedimientos) tendrán ámbito 1, 2, ... así sucesivamente siguiendo el orden en que se han definido las funciones (o procedimientos). Hay que señalar que la definición de las variables del programa principal puede hacerse antes o después de las funciones (o procedimientos) pero, si la declaración de las variables del programa principal se realizan después de las declaraciones de las funciones (o procedimientos), éstas pierden automáticamente el acceso a dichas variables. RECORDAR QUE SOLO SE PUEDE ACCEDER A ELEMENTOS YA DEFINIDOS.

Para manipular la TS y realizar las comprobaciones semánticas, necesitamos implementar las funciones de inserción y búsqueda sobre la TS.

```
void insertar_simb(...);
   //inserta un simb. en la TS, se le puede pasar directamente una estructura de tipo símbolo o
   bien la podéis implementar pasándole toda la información del símbolo para que ella lo cree y
   lo rellene con esa información.
SIMB_TAB buscar_simb(....);
   //se le pasa el nombre del símbolo a buscar y el ámbito y lo devuelve. Si el símbolo no
   existe devolvería un símbolo vacío o una indicación de error.
void imprimir_tabla_simbolos(void); //imprime la TS
```

¿Cómo se rellena la TS?

La tabla de símbolos se va rellenando conforme van definiéndose variables y funciones a lo largo del programa. Vamos insertando símbolos mediante instrucciones de código C (encerrado entre llaves {}), en determinados sitios de las producciones. Para ello debéis analizar **MINUCIOSAMENTE** la gramática y determinar sobre papel cuáles son estos sitios, y después se implementa.

Por ejemplo: Para introducir las variables.

Nos preguntamos, ¿Dónde se definen variables ?

Para definir el ámbito podéis usar una variable global que se incrementa casa vez que os metéis en un procedimiento

```
SIMB_TAB simb; // variable global
int ambito = 0; // variable global
declaration_list_opt → declaration_list | ε

declaration_list → declaration declaration_list_opt

declaration → TKN_CONST constant_declaration_list_opt |
              TKN_VAR variable_declaration_list_opt |
              procedure_declaration TKN_PTOCOMA

variable_declaration_list_opt → variable_declaration_list | ε
variable_declaration_list → variable_declaration variable_declaration_list_opt
variable_declaration → TKN_ID
                       {n=1; simb.tipo_simb= TIPO_VAR; strcpy(simb.nombre,$1);
                       simb.ambito = ambito; simb.tipo_dato=-1; insertar(simb);}
                       ident_list TKN_DOSPTOS type {rellenatipo($5) para los n+1
                       simbolos ultimos introducidos } TKN_PTOCOMA

type →      TKN_INTEGER {$$ = TIPO_INT}|
            TKN_REAL {$$ = TIPO_REAL}|
            TKN_BOOLEAN {$$ = TIPO_BOOL}

ident_list →TKN_COMA TKN_ID
            {simb.tipo_simb = TIPO_VAR; strcpy(simb.nombre,$2); simb.ambito = ambito;
                  simb.tipo_dato = -1; insertar(simb); n++} ident_list | ε
```

Algo similar habría que hacer para insertar en la TS las funciones y la información de sus argumentos.

¿Cómo se realizan las comprobaciones semánticas?

Igualmente las comprobaciones semánticas se realizan insertando instrucciones de código C (encerrado entre llaves {}), en determinados sitios de las producciones.

Por ejemplo: Para la comprobación de que la parte izquierda de una asignación no puede ser una función.

```
statement → TKN_ID {buscar que exite el simbolo y esta definido como variable, sino
            error } sentencia | if_statement | while_statement | repeat_statement  |
            for_statement | TKN_RETURN expression_opt
expression_opt → expression | ε
sentencia  → assignment | actual_parameters_opt
assignment → TKN_ASIGNACION expression
```

Por ejemplo: Para la comprobación de tipos en las expresiones.

```
term → factor {$$ = $1;} |
       term mul_operator factor {
         if (($1==TIPO_BOOL || ($3==TIPO_BOOL))
           printf("Error Semántico: operador no definido para tipo bool");
         else if ($1!=$3)
             printf("Error Semántico: operandos tienen tipos distintos");
       }
```

Nota:

- La tabla de símbolos como se utiliza en muchas partes del programa se puede definir como variable global. También se pueden utilizar variables auxiliares para ir recogiendo información sobre el tipo de dato, ámbito, etc, conforme nos movemos dentro de las producciones durante el reconocimiento.
- La TS y sus funciones de manipulación se deberían implementar en un modulo independiente, para después enlazar todos los módulos.
- Con esta información se trata solo de guiaros para implementar las comprobaciones semánticas, no hay que entenderla como una obligación. Cada uno puede implementar la estructura de datos y las funciones para manipularlas como quiera.