Universidad de La Laguna

E. T. S. DE INGENIERÍA INFORMÁTICA
Departamento de E.I.O. y Computación
Compiladores

Diseño e implementación de una Tabla de Símbolos para implementar el análisis de ámbito en el compilador de *Pascal-*

Factor de ponderación [0-10]: 10

8.1. Objetivos

La práctica consiste en ampliar el analizador sintáctico descendente recursivo con recuperación de errores de *Pascal-*, para que compruebe un conjunto de reglas que permitan al compilador asociar cada referencia a un identificador con una declaración del mismo.

Las reglas de ámbito de *Pascal-* son:

- **Regla 1:** Todas las constantes, tipos, variables y procedimientos definidos en el mismo bloque deben tener identificadores diferentes.
- **Regla 2:** Se puede hacer referencia a una constante, un tipo o una variable definida en un bloque desde el final de su declaración hasta el final del bloque. Se puede hacer referencia a un procedimiento definido en un bloque B desde el comienzo de la definición del procedimiento hasta el final del bloque B.
- **Regla 3:** Consideremos un bloque Q en el que está definido un identificador x. Si Q contiene a un bloque R en el que está definido otro identificador llamado x, el primer identificador es desconocido en el ámbito de la segunda declaración.

8.2. Descripción

Un programador de *Pascal-* utilizará identificadores para referirse a las diferentes entidades (variables, procedimientos, tipos, etc.) que componen el programa.

Cualquier identificador utilizado en un programa debe estar definido mediante una declaración en la que se definen algunos de sus atributos.

Algunos identificadores pueden utilizarse en cualquier programa *Pascal-* sin necesidad de declararlos. A estos identificadores los llamaremos identificadores estándar del lenguaje.

Un programa *Pascal-* combinará las declaraciones y las sentencias en unidades sintácticas denominadas Bloques (Pascal es un lenguaje con estructura de bloques).

Se distinguirán tres tipos de bloques:

- El bloque Estándar.
- El bloque del Programa
- Los bloques de Procedimientos.

```
{ Standard Block:
const   false = ...;
    true  = ...;
type    integer =...;
    boolean = ...;

procedure read(var x: integer);
begin   ...    end;

procedure write(x: integer);
begin   ...    end;

program test;
...
type  T = array[1..100] of integer;
var   x: T;

  procedure Q(x: integer);
  const c = 13;
  ...
    procedure R;
    var b, c: boolean;
    begin  ... x ... end;{ R }
  ...
  begin  ... x ...  end; { Q }

  procedure S;
  begin  ... x ...  end; { S }

  begin  ...  end.  { test }
}
```

Código 8.1: Estructura anidada de bloques

El bloque estándar es un bloque imaginario en el que estarán definidos los identificadores estándar. Puesto que *Pascal-* es un subconjunto de Pascal se permitirá el anidamiento de procedimientos, por lo que un bloque podrá contener a otro. El Código 8.1 muestra una estructura anidada de bloques de *Pascal-*. El bloque estándar contiene al programa completo.

El bloque del programa contiene a los procedimientos. Cada procedimiento puede a su vez contener a otros procedimientos. El propósito de los bloques es restringir el uso de un identificador al bloque en el que está definido (respondiendo al paradigma de encapsulamiento de la información).

A cada bloque de un programa se le asociará un nivel del siguiente modo: el bloque estándar tendrá nivel 0. El nivel se incrementará en uno al comienzo de cada bloque y se decrementará en uno a la salida del bloque, tal como muestra el Cuadro 8.1.

Bloque	Nivel
Standard Block	0
Program Block	1
Procedure Q	2
Procedure R	3
Procedure S	2

Cuadro 8.1: Nivel correspondiente a cada bloque del Código 8.1

Cada registro de la Tabla de Símbolos (TS) describe a un identificador mediante su lexema (*Name*).

En un programa encontraremos diferentes clases de identificadores: constantes, tipos, variables, procedimientos, etc. y los atributos que se asocian con cada identificador son diferentes dependiendo de la *clase* del identificador. Por ejemplo: una constante estará descrita por su tipo y su valor, mientras que un procedimiento se caracteriza por su lista de parámetros. Usaremos nombres como los que se muestran en el Código 8.2 para clasificar los identificadores en las diferentes clases que éstos pueden tener (el prefijo `CL_` indica *clase*).

```
typedef enum {CL_ARRAY_TYPE, CL_CONSTANT, CL_FIELD, CL_PROCEDURE,
          CL_RECORD_TYPE, CL_STANDARD_PROC, CL_STANDARD_TYPE,
          CL_VALUE_PARAMETER, CL_VAR_PARAMETER, CL_VARIABLE,
          CL_UNDEFINED
} class;
```

Código 8.2: Los diferentes tipos de entidades en el programa

El compilador debe asociar a un identificador todos los atributos que suministre su declaración, y esta asociación se hace a través de la Tabla de Símbolos (TS).

La primera decisión a tomar es la implementación de TS que utilizará el compilador. La figura 8.1 es una representación gráfica del estado de la TS hasta el nivel 3 para el Código 8.1. En la figura 8.1 se supone que la TS se ha implementado como una pila de listas enlazadas (se incluye esta figura a título de ejemplo, pero la implementación **no tiene porqué ser esa**).

El Código 8.3, escrito en C muestra el conjunto de atributos que se asocia con cada identificador en la TS, dependiendo de su clase. De todos estos atributos, en esta práctica sólo colocaremos el atributo `kind` (clase) del identificador.

```
typedef struct ObjectRecord * ptr;

typedef struct ObjectRecord {
  int name;
  ptr previous;
  class kind;

  union {
      /* CL_ARRAY_TYPE */
      struct {
         int LowerBound,
       UpperBound;
         ptr IndexType,
       ElementType;
      } AS; /* Array Struct */
      /* CL_CONSTANT */
      struct {
         int ConstValue;
         ptr ConstType;
      } CS; /* Constant Struct */
      ptr FieldType; /* CL_FIELD */
      ptr LastParam; /* CL_PROCEDURE */
      ptr LastField; /* CL_RECORD_TYPE */
   ptr VarType; /* CL_VALUE_PARAMETER CL_VAR_PARAMETER CL_VARIABLE */
  } KU; /* Kind Union */
} ObjectRecord;
```

Código 8.3: Atributos asociados con los identificadores

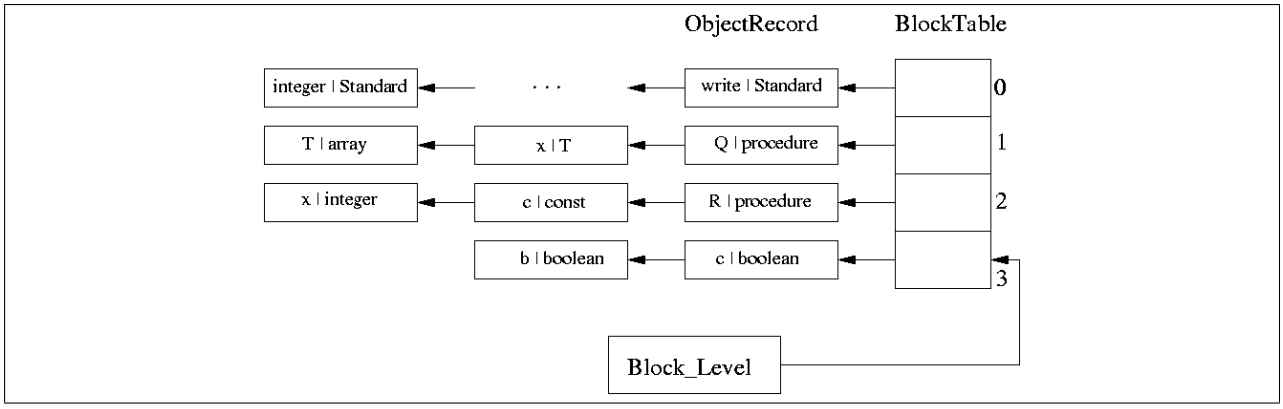

Figura 8.1: Estado de la pila de tablas de símbolos para el programa de la Figura 8.1

Para especificar la forma en la que se ha de realizar la recuperación de errores consideremos el ejemplo del Código 8.4.

Si se da esta situación, a menos que se tome alguna medida, el compilador

emitirá por cada aparición del identificador *table* un mensaje en el que se indique que hay un identificador no definido. Para que esto no ocurra, se utilizará la siguiente fórmula de recuperación de errores:

Si un identificador no aparece definido en la TS, se debe informar al usuario de que es un identificador no definido. El identificador se añadirá a la TS para evitar que cualquier otra referencia al mismo identificador sea la causante de mensajes de error adicionales.

Cuando el compilador analice la declación de un identificador en un bloque, comparará el nuevo identificador con los que ya han sido definidos en el mismo bloque. Si uno de los identificadores de la tabla coincide con el que se pretende insertar, se informará al usuario de que se trata de un "identificador ambiguo" y no se introduce en el ámbito correspondiente. Además como ya está en la TS, cualquier otra referencia a ese identificador no generará más mensajes de error.

```
program P;
...
var  tabel: T;
begin
  ... table ...
  ... table ...
  ... table ...
end.
```

Código 8.4: Un identificador no definido

Se enumeran a continuación los pasos que se han de desarrollar para la realización de esta práctica:

1. Escribir una función que determine si un identificador se ha declarado en un determinado bloque (el bloque se identificará por su nivel). Si el identificador está en la TS correspondiente a ese nivel, se devolverá un puntero al mismo:

   ```
   void search(...);
   ```

   Esta función *search()* es diferente de la que se utilizó en el módulo de análisis léxico (que tenía el mismo nombre). La función busca un identificador en la subtabla de símbolos local al bloque en curso de compilación (búsqueda local).

   search() sirve de base a algunas de las operaciones que tienen que llevar a cabo otras funciones que se describen a continuación.

2. Escribir una función para definir un nuevo identificador en el bloque en curso. Dicha función ha de realizar las siguientes acciones:

   - Determinar si el identificador ya se ha utilizado en otra declaración en el mismo bloque.
   - Si ha sido declarado previamente, se ha de emitir un mensaje de error en el que se indique la presencia de un identificador ambiguo.

- En cualquier caso, se ha de insertar el identificador en la TS correspondiente al bloque; devolviendo un puntero al identificador. El puntero que se devuelve se utilizará (en las siguientes prácticas) para rellenar atributos adicionales del mismo identificador:

```
void insert(...);
```

  Esta función *insert()* también es diferente de la que se utilizó en el módulo de análisis léxico (y que tenía el mismo nombre). Si el identificador es encontrado, se notifica un eror: ”Id. ambiguo”.

- Escribir una función que valide una referencia a un identificador.

  Cuando se hace referencia a un identificador en un bloque, se ha de buscar la definición que le precede, en el mismo bloque. Si esta búsqueda falla, entonces se debe buscar en las definiciones del bloque inmediato superior y así sucesivamente hasta que se encuentre la definición correspondiente o se alcance el bloque estándar. Este proceso se puede resumir como:

  - Paso 1: Si una referencia a un identificador x está precedido en un mismo bloque por la declaración de un identificador con ese lexema (x), entonces la referencia a x se refiere a ese identificador (al declarado).
  - Paso 2: En otro caso, repetir el Paso 1 en el bloque inmediato superior que contiene al bloque en el que se acaba de buscar.

  Si el identificador no aparece definido en ninguno de los bloques se debe emitir un mensaje que indique que el identificador no está definido e insertarlo con el atributo no definido (*undefined*).

```
void lookup(...);
```

- Escribir una función para inicializar la TS correspondiente a cada nuevo bloque:

```
void set(...);
```

- Escribir una función para “desactivar” la TS correspondiente a un bloque:

```
void reset(...);
```

- Definir los identificadores estándar a nivel cero. Utilizar para ello una función similar a la que se muestra en el Código 8.5. En ese código `NO_NAME` es un número que corresponderá con un identificador “inexistente”: cuando se espera encontrar un identificador (*match()*), si se encuentra vendrá acompañado del atributo *name*, que será un número (único) identificativo del identificador en cuestión. Si no se encuentra el identificador, *match()* falla y el atributo *name* se hace igual al valor `NO_NAME` para implementar la recuperación de errores.

  Se debe realizar una llamada a esta función antes de comenzar el análisis sintáctico.

- Revisar las funciones del analizador sintáctico en las que se hacen declaraciones de identificadores e invocar a la función *insert()* para incluir en la tabla los identificadores definidos.

```
void StandardBlock(void) {
     ptr obj;
     /* Inicializamos todos los bloques a vacio */
     for (BlockLevel = 0; BlockLevel <= MAX_LEVEL; BlockLevel
        ++)
   BlockTable[BlockLevel].LastObject = NULL;

     /* Inicializamos el bloque cero */
     BlockLevel = -1;
     set();
     insert(NO_NAME, CL_STANDARD_TYPE, &obj);
     insert(INTEGER, CL_STANDARD_TYPE, &obj);
     insert(BOOLEAN, CL_STANDARD_TYPE, &obj);
     insert(FALSE0, CL_CONSTANT, &obj);
     insert(TRUE0, CL_CONSTANT, &obj);
     insert(READ, CL_STANDARD_PROC, &obj);
     insert(WRITE, CL_STANDARD_PROC, &obj);
}
```

Código 8.5: Definición de los identificadores estándar

  Puede suceder que en el análisis de una declaración no se encuentre el identificador correspondiente al identificador que se desea definir. Esto representa un error sintáctico y se debe tener en cuenta a la hora de insertar el elemento en la TS. Lo que se debe hacer realizar la inserción usando `name = NO_NAME`.

- Revisar las funciones del analizador sintáctico en las que se hace referencia a un identificador y en ellas comprobar que efectivamente se ha declarado un identificador (lookup).
- Al entrar y salir de un bloque (block_body) se tienen que activar y desactivar de manera conveniente las tablas de símbolos correspondientes (*set* y *reset*).

Una vez diseñadas todas las funcionalidades que se han descrito, preparar el analizador para que admita en línea de comandos un nombre de fichero adicional el el que volcará unas estadísticas sobre el acceso a la TS. Por ejemplo, si el nombre del analizador es `p-c` se podrá ejecutar como: `p-c program.p` (para analizar el programa `program.p`) o bien como `p-c program.p estad.txt`, volcando las estadísticas de trabajo con la TS en el fichero `estad.txt`.

Las estadísticas que se volcarán contemplarán al menos la siguiente información:

- Tamaño de la TS
- Número total de identificadores declarados en el programa.

- Número total de identificadores referenciados.
- Número de colisiones en la TS
- Longitud máxima de las listas de encadenamiento en la TS

Aparte de las estadísticas, el compilador deberá imprimir en pantalla el tiempo que invierte en compilar un fichero con código fuente de *Pascal-*. Para ello, busque información sobre funciones de bajo nivel (sistema) que permitan realizar medidas de tiempo.

8.3. Notas

Debe estudiarse el comportamiento del programa desarrollado con los todos los ficheros fuentes de *Pascal-* que hayan codificado hasta este momento.

Asimismo se puede incorporar a las estadísticas cualquier información adicional que consideren de interés.