Bitiondo - Etapa III

Tabla de Símbolos y Chequeo de Tipos (8%)

Especificación de la entrega

Hasta este momento hemos construido un analizador lexicográfico y sintáctico para el lenguaje de programación `Bitiondo`. Esta siguiente etapa del desarrollo corresponde a la verificación del uso correcto de tipos dentro del lenguaje además de la implementación y uso de una **Tabla de Símbolos**.

Para este punto, su interpretador de código `bitiondo` debe de reportar -si existen- los siguientes errores:

- Redeclaración de variables dentro de un mismo bloque.
- Utilización de variables no declaradas en ese alcance.
- Modificación de variables de iteración.
- Errores de tipos, como por ejemplo: Intentar sumar una variable del tipo `int` con una del tipo `bool`.

Para ello deberán implementar y *llenar* una **Tabla de Símbolos**:

Estructura de datos que posee información de declaración y contexto de cada identificador incluido en un código en `Bitiondo`. Recuerden que a pesar de no ser declaradas al inicio de un bloque, las variables de iteración son **también** tomadas en cuenta para la tabla de símbolos. La implementación debe estar en un módulo o archivo aparte y **debe incluir** como mínimo los siguientes procedimientos (si está programando en castellano, puede colocar los equivalentes):

- `SymTable.new()`: Construye una tabla de símbolos vacía.
- `SymTable.insert(...)`: Inserta un elemento a la tabla de símbolos.
- `SymTable.delete(...)`: Elimina un elemento de la tabla de símbolos.
- `SymTable.update(...)`: Actualizala información de un elemento de la tabla de símbolos.
- `SymTable.isMember(...)`: Determina si un elemento se encuentra en la tabla de símbolos.
- `SymTable.find(...)`: Retorna la información de un elemento en la tabla de símbolos suponiendo que dicho elemento existe.

Los parámetros de cada método se dejan a preferencia y conveniencia del equipo. La implementación de cada uno de estos métodos será tomada en cuenta al momento de la corrección.

Un elemento en la tabla de símbolos debe tener la siguiente información:

- Nombre.
- Tipo.
- Valor.

- Tamaño. (Sólo en caso del tipo `bits`).

Ejecución

Para la ejecución del interpretador su programa deberá llamarse `bitiondo` y recibirá como primer argumento el nombre del archivo con el código en `Bitiondo` a analizar.

Luego de realizar el análisis lexicográfico y sintáctico, de no presentarse errores relacionados con estos, se procede a verificar errores de alcance, redeclaración y no declaración utilizando la tabla de símbolos. Finalmente se chequea la correctitud en el uso de tipos.

En caso de encontrar errores lexicográficos y sintácticos, deben reportar **sólo** estos de la misma manera que en entregas anteriores. Los errores relacionados a la verificación haciendo uso de la tabla de símbolos y errores de mal utilización de tipos deberán ser reportados **juntos**.

Por salida, se debe mostrar el árbol sintáctico abstracto que se ha desarrollado en la etapa anterior sustituyendo las instrucciones de declaración por la tabla de símbolos al inicio de cada nuevo alcance. En caso de encontrar errores, sólo deben imprimirse los errores.

El formato de impresión debe ser similar al ejemplo mostrado a continuación:

Programa:

```
begin
    int i;
    int j;
    bool b;
    bits s[4];

    if(true)
        begin
            bits i[4];
            bool b;

            i = 0b1111;
            forbits i as i from j going higher
                begin
                    bool i;
                end
        end
end
```

Salida correspondiente:

```
BEGIN
```

```
    SYMBOL TABLE
        Name: i, Type: int
        Name: j, Type: int
        Name: b, Type: bool
        Name: s, Type: bits, Size: 4

    IF
        condition:
            const_bool: true
        instruction:
            BLOCK
                SYMBOL TABLE
                    Name: i, Type: bits, Size: 4
                    Name: b, Type: bool

                ASSIGN
                    variable: i
                    value: const_bits: 0b1111

                FORBITS
                    bits:
                        variable: i
                    iteration:
                        variable: i
                    from:
                        variable: j
                    going: higher
                    instruction:
                        SYMBOL TABLE
                            Name: i, Type: int
                        BLOCK
                            SYMBOL TABLE
                                Name: i, Type: bool
END
```

Es de notar que debido a las reglas de alcance de `Bitiondo`, una variable puede *esconder* a otra de un alcance superior. Esto se observa en el segmento de código:

```
forbits i as i from j going higher
    begin
        bool i;
    end
```

En este caso, la `i` a la izquierda del `as` se encuentra en la tabla de símbolos del bloque que contiene a la instrucción `forbits`. La variable que está a la derecha del `as` es la variable de iteración y se encuentra en la tabla de símbolos de la instrucción `forbits`. Finalmente, la variable `i` de tipo `bool` declarada dentro

del bloque pertenece a la tabla de símbolos de dicho bloque.

Análogamente un caso con errores debe seguir el formato:

Redeclaración de variables en el mismo alcance:

```
begin
    int i;
    int i;
end
```

Salida correspondiente:

```
Error en línea 2, columna 9: La variable 'i' ya ha sido declarada en este alcance.
```

Errores de chequeo de tipos:

```
begin
    int i;
    i = 10;
    bool b = i && false;

    if (i + 2)
        outputln "Berro...";
end
```

Salida correspondiente:

```
Error en línea 4, columna 14: Operador '&&' no puede funcionar con expresiones de tipo 'int'
Error en línea 6, columna 9: Instrucción 'if' espera expresión de tipo 'bool'.
```

Errores de modificación de variable de iteración:

```
begin
    for (b = 0; b < 3; 1)
        b = 2;
end
```

Salida correspondiente:

```
Error en línea 3, columna 9: No es posible modificar la variable de iteración 'b'.
```

Implementación y formato de entrega

La implementación y el formato de entrega se mantienen iguales a las etapas anteriores.

Fecha de entrega:

La fecha límite de entrega del proyecto es el día **miércoles 22** de noviembre de 2017 (semana 10) *hasta* las **11:50pm**. Entregas hechas más tarde tendrán una **penalización del 20%** de la nota. Esta penalización aplica por cada día de retraso.