Acciones semánticas del compilador

Notación intermedia: Árbol

```
S:
    {
        printf("\n\nInicia el COMPILADOR\n\n");
    }
    prog {
        SPtr = ProgPtr;
        programa = SPtr;

        generarAssembler();
        guardarTS();
        printf("Regla 0\n");
        printf("\nCompilacion OK.\n");
    };
```

Regla 0: En esta regla se asignarán los punteros de la raíz del árbol.

Como es la última regla que se ejecutara es el momento ideal para escribir la tabla de símbolos obtenida y generar el código assembler a través del recorrido del árbol.

```
prog:
    sent { printf("Regla 1\n");
           ProgPtr = SentPtr;
         }
    |
    prog sent { printf("Regla 2\n");
                ProgPtr = crear_nodo(&num_nodo, "SENT", NODO_SIN_TIPO, ProgPtr, SentPtr);
              }
    ;
```

Regla 1: solo realizaremos una copia de punteros como indica la teoría.

Regla 2: Nuestro programa siempre comenzara con una sentencia, por lo que crearemos en esta sección el primer nodo del árbol. Al ser una regla recursiva, podremos tener la cantidad de sentencias anidadas que queramos.

```
sent:
    read { printf("Regla 3\n");
           SentPtr = ReadPtr;
         }
    |
    write { printf("Regla 4\n");
            SentPtr = WritePtr;
          }
    |
    asig { printf("Regla 5\n");
           SentPtr = AsigPtr;
         }
    ;
```

Reglas 3, 4 y 5: solo realizaremos una copia de punteros como indica la teoría.

```
read:
    READ ID { printf("Regla 6\n");
              char *valor = (char*) malloc(sizeof(char)*200);
              sprintf(valor,"%s",$2);
              valor[strlen(valor)] = '\0';
              IdCompPtr = crear_nodo(&num_nodo, valor, TIPO_INT, NULL, NULL);

              //lado derecho ";"

              //ifder

              AuxSPtr = crear_nodo(&num_nodo, "@esMenor", AUX, NULL, NULL);

              WritePtr = crear_nodo(&num_nodo, "WRITE", NODO_SIN_TIPO, AuxSPtr, NULL);
              FinPtr = crear_nodo(&num_nodo, "@fin", AUX, NULL, NULL);
              BiPtr = crear_nodo(&num_nodo, "BF", NODO_SIN_TIPO, WritePtr, FinPtr);

              //ifizq
              IdPtr = crear_nodo(&num_nodo, valor, TIPO_INT, NULL, NULL);
              UnoPtr = crear_nodo(&num_nodo, "@1", AUX, NULL, NULL);
              BLEPtr = crear_nodo(&num_nodo, "BLE", NODO_SIN_TIPO, IdPtr, UnoPtr);

              //if padre
              IfPtr = crear_nodo(&num_nodo, "IF", NODO_SIN_TIPO, BLEPtr, BiPtr);

              //lado izquierdo ";"
              IdPtr = crear_nodo(&num_nodo, valor, TIPO_INT, NULL, NULL);
              ReadPtr = crear_nodo(&num_nodo, "READ", NODO_SIN_TIPO, IdPtr, NULL);

              //padre ";"
              ReadPtr = crear_nodo(&num_nodo, ";", NODO_SIN_TIPO, ReadPtr, IfPtr);
            }
    ;
```

Regla 6: este es un caso más complejo. En esta regla leeremos el id ingresado por el usuario para que sea buscado en la lista mediante el nodo READ. Como en este compilador, esta regla solo se utilizará para ingresar el Pívot a buscar

en la lista, aproveche la ocasión para realizar la validación correspondiente. El pívot debe ser mayor a 0. Para eso realizamos una comparación mediante un IF en donde se evaluará dicha condición con el nodo BLE (salta si es menor a 1). Si se da el caso, el ir tendrá como hijo derecho tendrá un mensaje de error y un salto incondicional al final del programa.

```
asig:
    ID ASIGNA posicion { printf("Regla 7\n");

                         SaltoPtr = crear_nodo(&num_nodo, "@saltoET", AUX, NULL, NULL);

                         char *valor = (char*) malloc(sizeof(char)*200);
                         sprintf(valor,"%s",$1);
                         obtenerID(valor);
                         limpiarString(valor,valor);
                         valor[strlen(valor)] = '\0';
                         IdPtr = crear_nodo(&num_nodo, valor, TIPO_INT, NULL, NULL);

                         PosPtr = crear_nodo(&num_nodo, "@pos", AUX, ListaPtr, SaltoPtr);

                         AsigPtr = crear_nodo(&num_nodo, "ASIGNA", NODO_SIN_TIPO, IdPtr, PosPtr);
                       }
    ;
```

Regla 7: esta regla será la encargada de Asignar la posición del pívot encontrado en la lista a la variable “resul”. También aprovecharemos el nodo @post para enganchar de su lado izquierdo todo el recorrido de la lista, y del lado derecho, una etiqueta a donde saltare en caso de encontrar la primera posición del pívot.

```
POSICION PARA ID PYC CA CC PARC { printf("Regla 9\n");

                                  AuxSPtr = crear_nodo(&num_nodo, "@vacia", AUX, NULL, NULL);

                                  WritePtr = crear_nodo(&num_nodo, "WRITE", NODO_SIN_TIPO, AuxSPtr, NULL);
                                  FinPtr = crear_nodo(&num_nodo, "@fin", AUX, NULL, NULL);

                                  ListaPtr = crear_nodo(&num_nodo, "BF", NODO_SIN_TIPO, WritePtr, FinPtr);
                                }
;
```

Regla 9: este es un caso especial, en el cual la lista puede venir vacía. Como el nodo @pos ya no tendrá de su lazo izquierda la lista de constantes quedara vacío, lo cual es conveniente para insertarle como hijo izquierdo el mensaje corresponte de “lista vacía” y un salto incondicional al final del programa.

```
lista:
    CTE { printf("Regla 10\n");
          char cad[20];

          //sumatoria del aux
          AuxPtr = crear_nodo(&num_nodo, "@aux", AUX, NULL, NULL);
          UnoPtr = crear_nodo(&num_nodo, "@1", AUX, NULL, NULL);
          SumPunt = crear_nodo(&num_nodo, "+", SUMA, AuxPtr, UnoPtr);

          //lado izquierdo (condicion)
          IfDerPtr = IdCompPtr;

          char *valor = (char*) malloc(sizeof(char)*200);
          itoa($1, cad, 10);
          sprintf(valor,"%s",cad);
          valor[strlen(valor)] = '\0';
          IfIzqPtr = crear_nodo(&num_nodo, valor, TIPO_INT, SumPunt, NULL);
          CmpPtr = crear_nodo(&num_nodo, "CMP", NODO_SIN_TIPO, IfIzqPtr, IfDerPtr);

          //lado derecho (guardar pos y salto)
          PosPtr = crear_nodo(&num_nodo, "@pos", AUX, NULL, NULL);
          AuxPtr = crear_nodo(&num_nodo, "@aux", AUX, NULL, NULL);
          AsigPtr = crear_nodo(&num_nodo, "ASIGNA", NODO_SIN_TIPO, PosPtr, AuxPtr);

          SaltoPtr = crear_nodo(&num_nodo, "@salto", AUX, NULL, NULL);

          BiPtr = crear_nodo(&num_nodo, "BI", NODO_SIN_TIPO, AsigPtr, SaltoPtr);

          //padre
          ListaPtr = crear_nodo(&num_nodo, "IF", NODO_SIN_TIPO, CmpPtr, BiPtr);
        }
```

Regla 10: esta será la regla más importante. Crearemos un nodo auxiliar con un “;” para poder trabajar mejor. Cada vez que entremos a un elemento de una lista incrementaremos la variable @aux, esto será útil para saber en qué posición de la lista nos encontramos. Luego mediante un IF preguntaremos si el pivot (puntero que tendremos de la regla6) es igual a elemento de la lista, de ser así, guardaremos el valor de @aux en @pos, que será nuestra variable auxiliar que asignaremos a nuestro id resul. De no ser iguales, saltaremos al siguiente elemento de la lista.

```
|
lista COMA CTE { printf("Regla 11\n");
                 char cad[20];

                 //sumatoria del aux
                 AuxPtr = crear_nodo(&num_nodo, "@aux", AUX, NULL, NULL);
                 UnoPtr = crear_nodo(&num_nodo, "@1", AUX, NULL, NULL);
                 SumPunt = crear_nodo(&num_nodo, "+", SUMA, AuxPtr, UnoPtr);

                 //lado izquierdo (condicion)
                 IfDerPtr = IdCompPtr;

                 char *valor = (char*) malloc(sizeof(char)*200);
                 itoa($3, cad, 10);
                 sprintf(valor,"%s",cad);
                 valor[strlen(valor)] = '\0';
                 IfIzqPtr = crear_nodo(&num_nodo, valor, TIPO_INT, SumPunt, NULL);
                 CmpPtr = crear_nodo(&num_nodo, "CMP", NODO_SIN_TIPO, IfIzqPtr, IfDerPtr);

                 //lado derecho (guardar pos y salto)
                 PosPtr = crear_nodo(&num_nodo, "@pos", AUX, NULL, NULL);
                 AuxPtr = crear_nodo(&num_nodo, "@aux", AUX, NULL, NULL);
                 AsigPtr = crear_nodo(&num_nodo, "ASIGNA", NODO_SIN_TIPO, PosPtr, AuxPtr);

                 SaltoPtr = crear_nodo(&num_nodo, "@salto", AUX, NULL, NULL);

                 BiPtr = crear_nodo(&num_nodo, "BI", NODO_SIN_TIPO, AsigPtr, SaltoPtr);

                 //padre
                 IfPtr = crear_nodo(&num_nodo, "IF", NODO_SIN_TIPO, CmpPtr, BiPtr);
                 ListaPtr = crear_nodo(&num_nodo, ";", NODO_SIN_TIPO, ListaPtr, IfPtr);
               }
;
```

Regla 11: ídem anterior, pero con recursividad

```
write:
    WRITE CTE_S { printf("Regla 12\n");
                  char *valor = (char*) malloc(sizeof(char)*200);
                  sprintf(valor,"%s",$2);
                  valor[strlen(valor)] = '\0';

                  CteSPtr = crear_nodo(&num_nodo, valor, TIPO_STRING, NULL, NULL);
                  WritePtr = crear_nodo(&num_nodo, "WRITE",  NODO_SIN_TIPO, CteSPtr, NULL);
                }
    |
```

Regla 12: el nodo WRITE tendrá solo un hijo izquierdo que será necesario para mostrar constantes strings por pantalla.

```
|
WRITE ID { printf("Regla 13\n");
           char *valor = (char*) malloc(sizeof(char)*200);
           sprintf(valor,"%s",$2);
           valor[strlen(valor)] = '\0';

           //derecha de ";"
           IdPtr = crear_nodo(&num_nodo, valor, TIPO_INT, NULL, NULL);
           WritePtr = crear_nodo(&num_nodo, "WRITE", NODO_SIN_TIPO, IdPtr, NULL);

           //izquierda de ";"
           AuxSPtr = crear_nodo(&num_nodo, "@perdido", AUX, NULL, NULL);
           WritePtrAux = crear_nodo(&num_nodo, "WRITE", NODO_SIN_TIPO, AuxSPtr, NULL);

           FinPtr = crear_nodo(&num_nodo, "@fin", AUX, NULL, NULL);
           BiPtr = crear_nodo(&num_nodo, "BF", NODO_SIN_TIPO, WritePtrAux, FinPtr);

           MenosUnoPtr = crear_nodo(&num_nodo, "@unoNeg", AUX, NULL, NULL);
           IdPtr = crear_nodo(&num_nodo, valor, TIPO_INT, NULL, NULL);
           CmpPtr = crear_nodo(&num_nodo, "CMP", NODO_SIN_TIPO, IdPtr, MenosUnoPtr);

           IfPtr = crear_nodo(&num_nodo, "IF", NODO_SIN_TIPO, CmpPtr, BiPtr);

           WritePtr = crear_nodo(&num_nodo, ";", NODO_SIN_TIPO, IfPtr, WritePtr);
         }
;
```

Regla 13: esta regla la utilizaremos para mostrar IDs. Como en este compilador esta regla solo se utiliza para mostrar la posición encontrada del pívot dentro de la lista, se puede aprovechar para realizar una validación. Si el Id resul contiene “-1”, que es el valor con el que inicializamos @pos, entonces significa que el pívot no fue encontrado, salta el mensaje correspondiente y saltamos al final del programa. De lo contrario se muestra la primera posición del pívot encontrado en la lista.