1.7. Traducción de funciones

En esta sección se describe en términos generales el mecanismo de generación de código de 3-direcciones para funciones. La generación de código para las funciones es la parte más complicada porque depende de la máquina objeto y de la organización del entorno de ejecución. La traducción de funciones implica dos etapas:

- **la definición de la función**. Una definición de función crea un nombre, parámetros y código del cuerpo de la función pero no se ejecuta la función en ese punto.
- **la llamada a la función** desde una parte del programa (trataremos a los procedimientos como funciones que no devuelven valores). Una llamada crea los valores actuales para los parámetros y realiza un salto al código de entrada de la función que se ejecuta y retorna al punto de la llamada.

Supongamos que el subárbol sintáctico correspondiente a una función tiene como raíz un nodo de tipo `n_call`, llamada a función, con una serie de hijos que corresponden a los argumentos, que en general, serán expresiones $E_1, \ldots, E_n$ a evaluar para obtener el valor actual de los n-argumentos.

A continuación se incluye una implantación para generar el código de 3-direcciones correspondiente a la llamada de la función en primer lugar. Posteriormente analizaremos el caso de la definición de la función. Supondremos que se han comprobado previamente las condiciones semánticas de la llamada (número y tipo de argumentos). Usando la misma notación que para la práctica 3, la llamada a una función de dos argumentos genera el siguiente código intermedio:

Llamada a la función

```
ASIGN_C  t0  147  // asigna la dirección de retorno 147 a t0 (la siguiente al CALL)
PUSH     t0  –    // apila la dirección de retorno
PUSH     t1  –    // apila el valor de t1, primer arg. de la función
PUSH     t2  –    // apila el valor de t2, segundo arg. de la función
CALL     X   –    // salta a la dirección X, entrada a la función
POP      t3  –    // direccion 147, se desapila el valor devuelto por la función, se almacena en t3
```

Cuadro 1.6: Ejemplo de código de llamada a una función

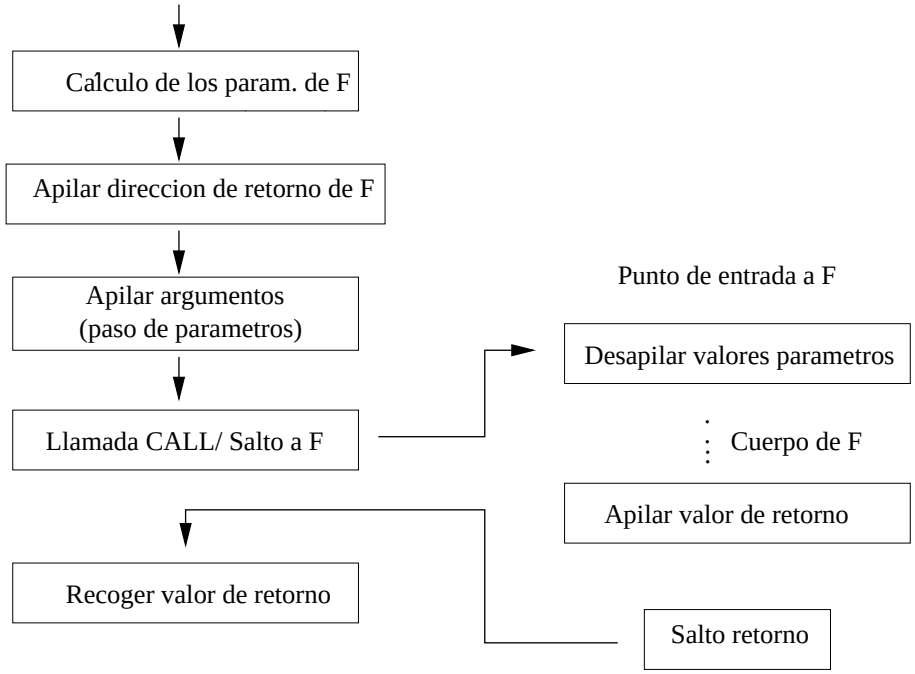

Figura 1.9: Diagrama de flujo para la llamada a función

Se hace uso de una pila en donde se apilan los valores actuales de los parámetros de la función para después, como veremos en el siguiente apartado, en la implantación del cuerpo de la función desapilarlos.

Nota: Respecto a la dirección de la llamada a la función (índice de la instrucción del código de entrada a la función, $dir2$ en el ejemplo anterior, figura 1.11), se podría conocer en el caso de que el código para la función ya se hubiera generado antes de la llamada a la función (por ejemplo, se podría haber almacenado en la entrada de la tabla de símbolos correspondiente a esa función en un campo definido para ese propósito).

En el caso de que aún no se haya generado el código para la definición de la función tendríamos que hacer un relleno de retroceso en todas las llamadas a esa función. Una forma sería para cada función almacenar los índices de las instrucciones donde se hace una llamada a dicha función (por ejemplo, en un vector de índices) y una vez definida la función que ya se conoce la dirección de entrada recorremos el vector de índices y rellenamos con la dirección adecuada en todas las proposiciones de llamadas a esa función.

```
TAC datos;
datos.cod=NULL;
TAC aux[MAXNCHILDS];
cuadruplos cod1, cod2, codpush[MAXNCHILDS], codcall, codpop;
direcciones dir1,dir2; //direcciones a los saltos
char *t0, *t1; // varibales temporales

case n_call:
   // cálculo de los parámetros de la función
   for(i = 0; i<nchilds;i++)
         aux[i] = genera_código(childs[i]);
   t0 = nuevotemp();
   //asigna en t0 la direc. de retorno de la func., aún no se sabe
   cod1 = gen_cuad(ASSIGN,t0,dir1?);
   cod2 = gen_cuad(PUSH,–,t0,–); // apilamos la direc. de retorno
   // apilamos los argumentos
   for(i = 0; i<nchilds;i++)
         codpush[i] = gen_cuad(PUSH,–, aux[i].lugar,–));
   // llamada a la func., no se sabe aún la direcc. de su cuerpo
   codcall = gen_cuad(CALL, –, dir2?,–);
   // la siguiente dirección libre es la de retorno
   dir1 = sigtedirlibre();
   rellena(cod1,arg3,dir1);
   t1 = nuevotemp();
   // recogemos el valor devuelto por la funcion
   codpop = gen_cuad(POP,t1,–,–) ;
   datos.cod=concatena(losaux[i].cod,cod1,cod2,loscodpush[i],codcall,codpop);
   datos.lugar = t1;
   //para relleno de retroceso de dir2, vease Nota
   return (datos);
```

Figura 1.10: Implantación de la llamada a funciones

Cuerpo de la función

Para traducir el cuerpo de la función bastaría recorrer el árbol hasta encontrar un nodo de tipo `n_function` a partir del cual colgará el bloque de sentencias del cuerpo de la función. No haría falta generar la parte del árbol de los argumentos ya que éstos estarían almacenados en la tabla de símbolos. A partir de cierta dirección de código intermedio deberíamos tener un instrucción que sería la entrada a la función y que contendría:

```
POP     t10   // recoge el segundo argumento
POP     t11   // recoge el primer argumento
...           // operaciones de la función que almacena el resultado en t20 (por ejemplo)
POP     t15   // recoge de la pila el valor de la direcc. de retorno (147) y lo pone en t15 (por ejem.)
RETURN  t15   t20 // salida de la función: el entorno de ejecución deber primero saltar a la instrucc.
              cuya dirección es el valor de t15 y apilar el valor devuelto que es el de t20
```

Esto implica las siguientes tareas (ver parte derecha de la figura 1.9) :

1. Recoger la dirección libre actual y utilizarla como dirección para rellenar (relleno con retroceso) las llamadas a la función que estamos definiendo.
2. Generar código para desapilar los argumentos de la función y meterlos en las variables correspondientes.
3. Generar código para el bloque de sentencias S.
4. Desapilar la direccion de retorno.
5. Generar la salida de la funcion con RETURN.
6. Unir los trozos de código para tenerlos en una única lista de cuádruplos.

```
TAC datos;
datos.cod=NULL;
TAC aux[MAXNCHILDS];
cuadruplos codpoparg[MAXNPARAM], codpopdirreturn, codreturn;
direcciones dir; //direccion de retorno
char *t; // varibale temporal
case n_function:
   dir = sigtedirlibre();
   rellena(todas llamadas a la funcion con esta dir);
   // desapilamos los argumentos
   for(i = 0; i < nparams ;i++)
          codpoparg[i] = gen_cuad(POP, param[i].lugar, -, -));
       //genera codigo para el cuerpo de la función (k intrucciones)
   //y calculamos el valor a devolver
   for(k = 0; k < nchilds; k++)
          aux[k] = genera_código(childs[k]);
   t = nuevotemp();
   //desapilamos la direccion de retorno de la función
   codpopdirreturn = gen_cuad(POP, t, -, -);
   //salida de la funcion: el entorno de ejecución salta a la
   //direcc. de retorno en t y se apila el valor devuelto
   codreturn = gen_cuad(RETURN, t, valor devuelto, -);
  datos.cod=concatena(loscodpoparg[i],losaux[k].cod,codpopdirreturn,codreturn);
   datos.lugar = valor devuelto;
   return (datos);
```

Figura 1.11: Implantación del cuerpo de las funciones

OJO en VUESTRA práctica 3

- En la definición de la función, si que vais a tener que colgar los argumentos en el árbol para saber cuantos parámetros teneis que desapilar cuando implementeis el cuerpo de la función, pues no teneis la tabla de simbolos con esa información.
- Para los procedimientos si que teneis un nodo n_CALL. Para las funciones tendreis que saber si en una instruccion, por ejemplo a =suma(3,2), corresponde a un vector o llamada a función. En el segundo caso tendreis que crearos a mano un nodo n_CALL. Para poder distinguir los dos casos podeis crear una especie de *pseudo-Tabla de Símbolos* donde sólo se almacenen los nombres de las funciones y hacer una búsqueda.

- Para hacer el relleno con retroceso en las llamadas a función de su direccion de entrada, variable $dir2$, una forma sería para cada función almacenar los índices de las instrucciones donde se hace una llamada a dicha función (por ejemplo, en un vector de índices) y una vez que ya se conozca la dirección de entrada recorremos el vector de índices y rellenamos con la dirección adecuada en todos los cuadruplos de llamadas a esa función.

Vamos a hacer el siguiente ejemplo

```
FUNCTION Suma (a AS INTEGER, b AS INTERGER) AS INTEGER
DIM total AS INTEGER
  total = a + b ;
  Suma = total ;
END FUNCTION
REM Bloque principal
DIM resultado AS INTEGER
resultado = Suma ( 2, 4*8) ;
PRINT resultado ;
END
```

Listado de cuadruplos

```
(MULT, t1, 4, 8)
(ASSIGN, t2, 7, NULL)
(PUSH, t2, NULL, NULL)
(PUSH, 2, NULL, NULL)
(PUSH, t1, NULL, NULL)
(CALL, 11, NULL, NULL)
(POP, t3, NULL, NULL)
(ASSIGN, resultado, t3, NULL)
(WRITE, resultado, NULL, NULL)
(HALT, NULL, NULL, NULL) // final del programa
(POP, b, NULL, NULL) // entrada a la función
(POP, a, NULL, NULL)
(ADD, t4, a, b)
(ASSIGN, total, t4, NULL)
(ASSIGN, Suma, total, NULL)
(POP, t5, NULL, NULL) // se desapila la dirección de retorno, la 7
(RETURN, t5, Suma, NULL) // el entorno de ejecución salta a la direccion
// en t5 (la 7) y apila el valor a devolver
```