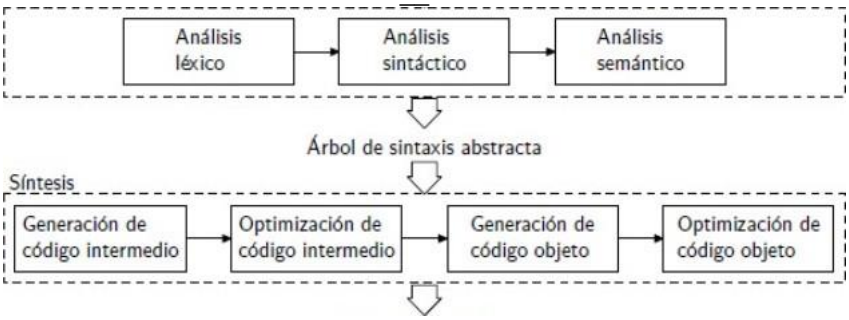
2.3 Esquema de generación

Los esquemas de generación son las estrategias o acciones que se deberán realizarse y tomarse en cuenta en el momento de generar código intermedio. Los esquemas de generación dependen de cada lenguaje. Tomaremos algunos esquemas de generación del lenguaje C.

▪ Esquema de generación de código



2.3.1 Variables y constantes

Declaración de variables y constantes.

Las declaraciones de variables y constantes deben separarse de tal manera que queden las expresiones una por una de manera simple.

▪ Por ejemplo int a,b,c;

se descompone a int a;

int b; intc; respectivamente.

Las variables utilizadas en los programas se clasifican en dos tipos:

variables locales y variables globales.

Variables locales:

Aquella que está declarada para el programa o algoritmo completo.

▪ Para definir variables locales, la definición debe hacerse inmediatamente después de una llave de inicio ({), y la variable deja de existir fuera de la llave de fin(}) que corresponde a la llave de inicio después del cuál fue definida la variable.

Ejemplo:

```
{
int a,b;
a=5;
b=a + 100;
}
```

Variables globales:

Aquella que está declarada y definida dentro de una función y sólo es válida dentro de la misma función y no podrá utilizarse en otra parte del programa.

▪ Una variable global se declara fuera de cualquier función y primero que cualquier función que requiera de ella. Una variable se declara de la siguiente forma:

▪ tipo identificador1, identificador2..ident n;

Ejemplos:

▪ Algunos ejemplos de cómo definir variables son los siguientes:

▪ int alumnos, contador;

▪ float A,B,C;

▪ char Letra;

Declaracion de Constantes.

Para declarar constantes en el lenguaje C, sólo basta anteponer el calificador const seguido del tipo de dato que manejará esa constante (int,char,etc), y por último el valor que tomará esa variable.

Ejemplo:

```
const int k = 12
```

2.3.2 Expresiones

En esta función recibe una cadena que representa una línea de código intermedio y toma las medidas oportunas para que ese código se utilice.

Estas medidas pueden ser escribir la línea en un fichero adecuado, almacenar la instrucción en una lista que después se pasará a otros módulos, o cualquier otra que necesitemos en nuestro compilador.

Expresiones aritméticas

Son aquella donde los operadores que intervienen en ella son numéricos, el resultado es un número y los operadores son aritméticos. Los operadores aritméticos más comúnmente utilizados son: +, - , * , / y %.

Comenzamos el estudio por las expresiones aritméticas. Lo que tendremos que hacer es crear por cada tipo de nodo un método que genere el código para calcular la expresión y lo emita. Ese código dejará el resultado en un registro, cuyo nombre devolverá el método como resultado.

Para reservar estos registros temporales, utilizaremos una función, reserva.

En principio bastar ‘a con que esta función devuelva un registro distinto cada vez que se la llame.

Cada nodo generará el código de la siguiente manera:

 Por cada uno de sus operandos, llamara al método correspondiente para que se evalúe la sub expresión. Si es necesario, reservara un registro para guardar su resultado.

 Emitirá las instrucciones necesarias para realizar el cálculo a partir de los operandos.

2.3.3 Instrucciones de asignación

La sintaxis general de la instrucción de asignación es: nombre_de_la_variable = valor

El valor a la derecha del signo igual puede ser una constante, otra variable o una expresión que combine constantes y variables, pero siempre la variable y su valor deben ser del mismo tipo de dato.

Ejemplos:

edad% = 5

area! = 12.3

nombre$ = “Pedro”

 Instrucciones de asignación compuesta

Las instrucciones de asignación compuesta realizan primero una operación en una expresión antes de asignarla a un elemento de programación. En el siguiente ejemplo se muestra uno de estos operadores, +=, que incrementa el valor de la variable del lado izquierdo del operador con el valor de la expresión de la derecha.

Una instrucción de asignación asigna el valor de una expresión a una variable. En general, si la variable que se va a asignar es una propiedad, la propiedad debe ser de lectura y escritura o de sólo escritura; en caso contrario, se produce un error de compilación. Si la variable es una variable de sólo lectura, la asignación debe producirse en un constructor Shared o un constructor de instancia apropiado para el tipo de la variable; en caso contrario, se producirá un error de compilación.

2.3.4 Instrucciones de control

Esta forma de programación sólo permite resolver problemas sencillos.

Para resolver problemas más complejos, nos puede interesar que, dependiendo de los valores de los datos, se ejecuten unas instrucciones u otras.

Las instrucciones condicionales nos van a permitir representar este tipo de comportamiento. Sentencias IF y SWITCH. En otros casos, nos encontraremos con la necesidad de repetir una instrucción o instrucciones un número determinado de veces. En estos casos utilizaremos instrucciones de control iterativas o repetitivas (ciclos). Sentencias WHILE, DO-WHILE y FOR.

En los lenguajes de programación hay estructuras y operadores que permiten controlar el flujo de la ejecución, estos pueden ser ciclos, saltos, condiciones entre otros.

Expresiones booleanas

En los lenguajes de programación, las expresiones booleanas tienen dos propósitos principales. Se utilizan para calcular valores lógicos y como expresiones condicionales en proposiciones que alteran el flujo del control, como las proposiciones if-else o do-while. Las expresiones booleanas se componen de los operadores boléanos (and, or y not) aplicados a los elementos que son variables booleanas o expresiones relacionales. Algunos lenguajes permiten expresiones más generales donde se pueden aplicar operadores booleanos, aritméticos y relacionales a expresiones de cualquier tipo, sin diferenciar valores booleanos de aritméticos; si es necesario se realiza una coerción.

Saltos

En el código de los saltos los operadores lógicos &&, || y! son traducidos a saltos, aunque estos no aparecen realmente en el código. Por ejemplo la expresión: if (x <100 || x > 200 && x!= y) x=0; se puede traducir como las siguientes instrucciones:

If x < 100 goto L2

If False x > 200 goto L1l

f False x != y goto L1

L2: x =0

L1: Si la expresión es verdadera se alcanza la etiqueta L2 y se realiza la asignación en 1 caso contrario no haría nada.

2.3.5 Funciones

Las funciones pueden reducir a en línea, lo que se hace es expandir el código original de la función. Las funciones se descomponen simplificando los parámetros de manera individual al igual que el valor de retorno.

Cuando se usan un arreglo como un argumento a la función, se pasa sólo la dirección del arreglo y no la copia del arreglo entero. Para fines prácticos podemos considerar el nombre del arreglo sin ningún índice como la dirección del arreglo.

Considerar el siguiente ejemplo en donde se pasa un arreglo a la función imp_rev, observar que no es necesario especificar la dimensión del arreglo cuando es un parámetro de la función.

```
void imp_rev(char s[])
    {
       int t;
       for( t=strlen(s)-1; t>=0; t--)
          printf("%c",s[t]);
    }

    main()
    {
       char nombre[]="Facultad";
       imp_rev(nombre);
    }
```

Función del Lenguaje, entendemos que es el uso de la lengua que hace un hablante. En simples palabras, las funciones del lenguaje son los diferentes objetivos, propósitos y servicio que se le da al lenguaje al comunicarse, dándose una función del lenguaje por cada factor que tiene éste, en donde la función que prevalece es el factor en donde más se pone énfasis al comunicarse. Diversos lingüistas (Karl Bühler,Roman Jakobson, Michael Halliday…) han propuesto

distintas clasificaciones de las funciones del lenguaje: Bühler propuso que existían únicamente tres funciones: La Representativa (por la cual se trasmiten informaciones objetivamente) La Expresiva o emotiva (que expresa sentimientos del emisor) La Conativa, mediante la que se influye en el receptor del mensaje a través de órdenes, mandatos o sugerencias.

2.3.6 Estructuras

El código intermedio no es el lenguaje de programación de ninguna máquina real, sino que corresponde a una máquina abstracta, que se debe de definir lo más general posible, de forma que sea posible traducir este código intermedio a cualquier máquina real. El objetivo del código intermedio es reducir el número de programas necesarios para construir traductores, y permitir más fácilmente la transportabilidad de unas máquinas a otras. Supóngase que se tienen n lenguajes, y se desea construir traductores entre ellos. Sería necesario construir n*(n-l) traductores.

Sin embargo, si se construye un lenguaje intermedio, tan sólo son necesarios 2*n traductores. Así por ejemplo un fabricante de compiladores puede construir un compilador para diferentes máquinas objeto con tan sólo cambiar las dos últimas fases de la tarea de síntesis.

Analizador sintáctico	Árbol stntácttco	Análisis semántico	Árbol stntácttco	de código	Códi intermedio	Generador de código	

Aunque un programa fuente se puede traducir directa mente al lenguaje objeto, algunas ventajas de utilizar una forma intermedia independiente de la máquina son:

- Se facilita la redestinación: se puede crear un compilador para una máquina distinta uniendo una etapa final para la nueva máquina a una etapa inicial ya existente.

- Se puede aplicar a la representación intermedia un optimizador de código independiente de la máquina.

BIBLIOGRAFÍA

Lenguaje y Autómatas II. Unidad VI Generación de Código Intermedio. M.C. Juan Carlos Olivares Rojas. pág. 1-36. Año 2010.
Generación de código procesador de lenguaje. Lenguaje y autómata II. Universidad técnica de México pág. 1-28.