



REPORTE PROYECTO FINAL



Diego Mora Delgado

COMPILADORES Y DESARROLLO DE LIBRERIAS 22/06/2018

Aspectos básicos del lenguaje

Se trata de un lenguaje débilmente tipado.

Para declarar variables solo hace falta poner el nombre y el valor de esta, por ejemplo, si se quiere declara e inicializar una variable:

```
|precio asigna 23.67;
```

Lo cual indica que se esta declarando la variable precio y se está inicializando con un valor de 23.67.

Para ejecutar repetidamente un bloque de código se utilizan los bucles.

Para declarar un bucle en el lenguaje se hace de la siguiente manera:

```
bucle(i=0,i<n);  
precio asigna 23.67;  
a suma (b,1);  
itera(i++,);
```

El bloque de código a ejecutar iterativamente comienza desde la declaración del bucle, hasta la modificación de la variable iteradora indicada por la palabra reservada itera.

Los condicionales se utilizan para evaluar una expresión y dependiendo de la evaluación lógica se decide si se ejecuta el bloque de código dentro del condicional.

La manera de declarar un condicional es la siguiente:

```
condicion(i<10,);  
a multiplica (a,a);  
fin;
```

El bloque de código dentro de la palabra reservada condición y fin, se trata del bloque que se ejecutara en caso que la condición sea verdadera.

Las funciones se declaran de la siguiente manera:

```
funcion corre;  
less resta(divisor,67);  
FIN;
```

Se declara la función mediante la palabra reservada función, seguida del nombre de la función, las líneas siguientes de código serán el bloque de código dentro de la función, hasta que se termine el cuerpo de la función con la palabra FIN.

Para mandar a llamar una función solo ahí que utilizar la palabra reservada llama seguida del nombre de la función:

```
mas suma(2,3);  
llama corre;
```

Definición de la gramática del lenguaje

Esta es la parte donde el lenguaje de programación y por consecuencia el compilador comienza a tomar forma.

G(VN,VT,S,P)

Donde:

VT :

```
this->operador.push_back("itera");
this->operador.push_back("bucle");
this->operador.push_back("asigna");
this->operador.push_back("multiplica");
this->operador.push_back("divide");
this->operador.push_back("suma");
this->operador.push_back("resta");
this->operador.push_back("condicion");
this->operador.push_back("fin");
this->operador.push_back("funcion");
```

```
this->alpha.push_back("a");
this->alpha.push_back("b");
this->alpha.push_back("c");
this->alpha.push_back("d");
this->alpha.push_back("e");
this->alpha.push_back("f");
this->alpha.push_back("g");
this->alpha.push_back("h");
this->alpha.push_back("i");
this->alpha.push_back("j");
this->alpha.push_back("k");
this->alpha.push_back("l");
this->alpha.push_back("m");
this->alpha.push_back("n");
this->alpha.push_back("o");
this->alpha.push_back("p");
this->alpha.push_back("q");
this->alpha.push_back("r");
this->alpha.push_back("t");
this->alpha.push_back("u");
this->alpha.push_back("v");
this->alpha.push_back("w");
this->alpha.push_back("x");
this->alpha.push_back("y");
this->alpha.push_back("z");
```

```
this->alpha.push_back("A");
this->alpha.push_back("B");
this->alpha.push_back("C");
this->alpha.push_back("D");
this->alpha.push_back("E");
this->alpha.push_back("F");
this->alpha.push_back("G");
this->alpha.push_back("H");
this->alpha.push_back("I");
this->alpha.push_back("j");
this->alpha.push_back("K");
this->alpha.push_back("L");
this->alpha.push_back("M");
this->alpha.push_back("N");
this->alpha.push_back("O");
this->alpha.push_back("P");
this->alpha.push_back("Q");
this->alpha.push_back("R");
this->alpha.push_back("S");
this->alpha.push_back("T");
this->alpha.push_back("U");
this->alpha.push_back("V");
this->alpha.push_back("W");
this->alpha.push_back("X");
this->alpha.push_back("Y");
this->alpha.push_back("Z");
```

```
this->numeros.push_back("0");
this->numeros.push_back("1");
this->numeros.push_back("2");
this->numeros.push_back("3");
this->numeros.push_back("4");
this->numeros.push_back("5");
this->numeros.push_back("6");
this->numeros.push_back("7");
this->numeros.push_back("8");
this->numeros.push_back("9");
```

VN:

{E}

S = E

P:

- $E \rightarrow E$;
- $E \rightarrow E$ función E
- $E \rightarrow E$ bucle E
- $E \rightarrow E$ condición E
- $E \rightarrow E$ itera E
- $E \rightarrow E$ multiplica E
- $E \rightarrow E$ resta E
- $E \rightarrow E$ divide E
- $E \rightarrow (E)$
- $E \rightarrow E$ asigna {0-9}
- $E \rightarrow \{a-z, A-Z, 0-9\}$

Generando un lenguaje

L{Expresiones con notación infix de dígitos, signos +,*,/,-,;, (,),|};

Dentro de la clase compilador el constructor se encarga de insertar la información dentro de las propiedades del objeto instanciado:

```
//El constructor inserta la gramatica a las propiedades del objeto instanciado
compilador();
~compilador(){}


```

Análisis léxico

Proceso de scanning


Para que el compilador comience con el proceso de compilación debe tener como entrada un archivo fuente, el cual contendrá las acciones y operaciones a computar.

En este caso dicho archivo fuente será un documento con extensión .txt.

 programa	14/06/2018 8:59	Documento de tex	1 KB
--	-----------------	------------------	------

El método getFuente() de la clase compilador se encarga de abrir y obtener la información contenida dentro de dicho archivo, este método regresa un vector de caracteres, el cual contiene la información del archivo fuente:

Texto contenido dentro del archivo:

 programa: Bloc de notas

Archivo Edición Formato Ver Ayuda

```
palabra contador asigna 4534;
flotante b;
flotante a = 45;
entero first = 79;

contador asigna contador + 1;
```

Al momento de mandar a llamar la función `getFuente()` se obtiene el siguiente vector de caracteres:

```
*****
palabra contador asigna 4534;flotante b;flotante a = 45;entero first = 79;contador asigna contador + 1;
*****
```

El vector utilizado para guardar esta cadena es dinámico.

El siguiente paso es generar los tokens, para ello identificamos los delimitadores de cada línea de código.

Esta tarea lo hace el método `setLineas()` perteneciente a la clase `compilador`, este método configura la propiedad líneas de la clase `compilador`:

```
c1->setLineas(fuente);
```

Una vez que se ha mandado a llamar este método se tienen las líneas dentro del vector de vectores de caracteres llamado líneas:

```
vector<string> lineas;
```

Si se quieren mostrar estas líneas existe un método perteneciente llamado `showLineas()`, el cual imprime por consola las líneas:

```
cout<<"*****"<<endl;
c1->showLineas();
cout<<"*****"<<endl;
```

```
*****
palabra contador asigna 4534
flotante b
flotante a = 45
entero first = 79
contador asigna contador + 1
*****
```

Existe un método llamado `getNlineas()` para obtener el número de líneas encontradas en el archivo fuente:

```
linesN = c1->getNlineas();
cout<<linesN<<endl;
```

Existe un método que te permite obtener el contenido de las líneas y almacenarlo en un vector de vectores de caracteres, esta función se llama `getConLines()`, el cual recibe como parámetro este vector e introduce la información en él.

```
void getConLines(vector <string>&);
```

Al momento de mandar a llamar esta función tendremos en nuestra función principal un vector de cadenas disponible para trabajar con él.

```
vector <string> lineas, reservadas;
```

```
c1->getConLines(lineas);  
for(int i = 0; i<lineasN; i++){  
    cout<<lineas[i]<<endl;  
}
```

Al ejecutar el código se obtiene el mismo resultado que con el método `showLineas()`, la diferencia es que ya se tiene en la función principal un paquete de datos para trabajar sin necesidad de estar llamando al objeto compilador para obtener las líneas o línea de código.

La clase compilador contiene un método para obtener la cantidad de palabras reservadas, además de mostrar las palabras reservadas del lenguaje.

```
//Obtiene La cantidad de palabras reservadas  
int getNres();  
//Muestra las palabras reservadas  
void showReservadas();  
  
cout<<"*****"<<endl;  
c1->showReservadas();  
cout<<c1->getNres()<<endl;  
cout<<"*****"<<endl;
```

El resultado obtenido por consola es:

```
*****  
entero  
flotante  
palabra  
funcion  
para  
si  
6  
*****
```

Existe otro método para obtener las palabras reservadas y almacenarlas en un vector, se trata del método `getReservadas()`.

```
vector<string> lineas, reservadas;

c1->getReservadas(reservadas);
for(int i = 0; i<nReservadas;i++){
    cout<<reservadas[i]<<endl;
}

....."*****".....
```

Gracias a este método ya se tendrán las palabras reservadas en la función principal.

Existe un método que ayuda a crear las propiedades de la línea, o configurar las partes de un token:

- Es reservada
- Es id
- Es numero
- Es operador

Esta función crea 4 listas dinámicas, en las cuales se almacenan las propiedades del archivo fuente al pasar por las funciones de escáner.

```
void compilador::setPropiedades(){
    this->lpropiedades.insert(clase("reservada",0));
    this->lpropiedades.insert(clase("identificador",0));
    this->lpropiedades.insert(clase("numero",0));
    this->lpropiedades.insert(clase("operador",0));
}
```

Mediante la cual se crea una lista asociativa de clave valor para cada elemento del token.

Si se quieren revisar los valores y propiedades de estas listas asociativas se puede utilizar la función, showlProp(), la cual imprime por consola estas listas:

```
*****
identificador      0
    numero         0
    operador       0
    reservada      0
```

Generación de tokens

En el proceso de generación de tokens se debe de analizar los componentes del archivo fuente, es decir, si alguna línea contiene palabras reservadas, números, identificadores y operadores, en este proceso solo se identifican estos grupos, aun sin definir la operación a computar o la acción a realizar.

Existen métodos dentro de la clase compilador para realizar el proceso de la generación de los tokens.

Uno de los métodos es el encargado de obtener los números encontrados dentro del vector de caracteres, el nombre de este método se llama esNumero(), este método recibe como parámetro un vector de caracteres y regresa el numero encontrado dentro del vector de caracteres.

```
//Esto metodo regresa el numero encontrado dentro de un vector de caracteres
int esNumero(string);
```


Por ejemplo, si este método recibe como entrada la siguiente cadena:

```
palabra contador asigna 4534
```

```
cout<<c1->esNumero(lineas[1])<<endl;
```

Al pasarle este vector de caracteres al método esNumero(), retorna la siguiente información:

```
4534
```

Por ejemplo, si ingresamos otra cadena que contenga una declaración con un identificador, por default las variables declaradas, pero no inicializadas se inicializaran en 0.

```
flotante b
```

Se obtiene la siguiente respuesta:

```
0
```

Lo cual indica que se esta declarando una variable sin inicializarla.

Otro método para la ayuda del proceso de generación de tokens es el llamado getTexto():

```
//Este metodo obtien todos los cracteres dentro del alfabeto {a-z} y {A-Z}  
string getTexto(string);
```

Este método regresa un vector de caracteres con todos los caracteres dentro de los conjuntos del alfabeto {a-z} y {A-Z}.

Por ejemplo, si pasamos como parámetro el siguiente vector de caracteres:

```
palabra contador asigna 4534
```

El método regresa el siguiente vector de caracteres:

```
palabracontadorasigna
```

En este punto podemos utilizar esta cadena sin simbolos y sin números para reconocer los operadores, identificadores y palabras reservadas.

El método esReservada(), toma un vector de caracteres como entrada este analiza dicho vector y determina si este contiene alguna palabra reservada, además por parámetro actualiza dos valores enteros indicando la posición dentro del arreglo donde fue encontrado el primer carácter de la palabra reservada y el ultimo.

Otra opción de método para solo obtener la palabra reservada es un método sobrecargado que solo recibe como parámetro un vector de caracteres, este actualiza el contador de tokens y regresa la palabra reservada encontrada.

```
//Este metodo revisa si en el vector de caracteres pasado como parametro existen palabras reservadas, si es el caso  
//regresa la posicion inicial y final dentro del vector de caracteres en donde fue encontrada dicha palabra  
string esReservada(string, int&,int&);  
//Este metodo sobrecargado solo indica si existe una palabra reservada en un vector de caracteres  
string esReservada(string);
```

Si pasamos como parámetro el vector de caracteres:

```
enterocontadorasigna
```

El vector de caracteres regresado por la función será:

```
entero
0
6
```

Indicando que la palabra reservada entero se encuentra desde la posición 0 a la 6.

Además de arrojar esa información el método también aumenta el contador de elementos del token.

```
identificador    1
    numero        4
    operador      1
    reservada     1
```

Indicando que en la presente línea se ha encontrado un identificador.

Existe otro método para encontrar los operadores, los cuales han sido previamente definidos en la gramática del lenguaje, este método se llama esOperador(), el cual recibe como parámetro un vector de caracteres y regresa al operador encontrado en el vector de caracteres pasado como parámetro, además contiene un método sobrecargado que actualiza dos variables de tipo entero indicando la posición de inicio y fin del operador.

```
//Este metodo regresa la palabra reservada encontrada en el vector de caracteres
//ademas actualiza dos variables enteras indicando la posicion del operador
string esOperador(string,int&,int&);
//Solo regresa el operador encontrado
string esOperador(string);
```

Por ejemplo, si se ingresa el vector de caracteres:

```
enterocontadorasigna
```

La respuesta de este método será:

```
asigna
14
6
```

Indicando que el operador encontrado es asigna, dentro de la posición 14 a 20.

Además, este método actualiza el contador de propiedades del token, indicando que se ha encontrado un operador.

```
identificador    1
    numero        4
    operador      1
    reservada     1
```

Existe un método para encontrar identificadores, llamado esIdentificador(), el cual recibe un vector de caracteres y regresa el identificador:

```
//Recibe un vector de caracteres y regresa el identificador  
string esIdentificador(string);
```

Por ejemplo, si recibe el vector de caracteres:

```
enterocontadorasigna
```

La salida de este método es:

```
contador
```

Indicando que la palabra contador es un identificador, este método incrementa las propiedades del token.

Existe un método que ofrece un sumario de la información del token generado a partir del análisis de 1 línea, este método es el showlProp(), el cual muestra por consola la cantidad de caracteres numéricos, identificadores, operadores y reservadas que se encontraron en la línea analizada.

```
//Muestra las propiedades del token generadas a partir del analisis de una linea  
void showlProp();
```

Después del análisis de la línea:

```
entero contador asigna 4534
```

Se obtuvieron las siguientes propiedades del proceso de creación de su token:

```
identificador      1  
    numero         4  
    operador       1  
    reservada      1  
-----
```

Indicando que en la línea se encontraron 1 identificador <contador>, 4 caracteres numéricos <4534>, 1 operador <asigna> y 1 palabra reservada <entero>.

Existe un método que utiliza todos estos métodos para procesar la línea de código, este método es llamado crearToken(), este método recibe como parámetros:

- 1) La línea de código a procesar
- 2) El objeto compilador
- 3) El vector donde se guardarán las palabras reservadas encontradas
- 4) El vector donde se guardarán los identificadores encontrados
- 5) El vector donde se guardarán los números encontrados
- 6) El vector donde se guardarán los operadores encontrados
- 7) Además, actualiza el valor de una bandera que indica si el numero encontrado es un flotante

```
bool entero = true; void showlProp();
```

```
//Este metodo crea y actualiza la informacion de los tokens generados, ademas de detectar si un numero es entero o flotante  
int crearToken(string,compilador*, vector<string>&, vector<string>&,vector<string>&,vector<string>&,bool&);
```

Por ejemplo, al pasar como parámetro la siguiente línea:

```
entero contador asigna 45.34
```

La información generada por la creación del token:

```
45.34
entero
asigna
contador
1
```

Indicando que el token contiene:

```
identificador      1
      numero      4
      operador      1
      reservada     1
```

1 identificador, 4 caracteres numéricos, 1 operador y 1 palabra reservada, además indicando que el número es flotante.

Al momento de ejecutar el generador de tokens para todo el archivo, la función generar tokens proporciona un sumario con la información de las líneas analizadas.

Por ejemplo, si analizamos las siguientes líneas de código:

```
contador asigna multiplica 45.34 5
buclepara b
a asigna 45
first asigna multiplica a b
```

El sumario contendrá la siguiente información:

```
identificador      4
      numero      7
      operador      5
      reservada     1
```

Se han detectado o mandado a llamar 4 identificadores, se han detectado 7 caracteres numéricos, se ha utilizado 5 operadores y 1 palabra reservada.

Análisis sintáctico y semántico

Existe una función de la clase compiladores que analiza los operadores de las líneas de código:

```
//Esta funcion revisa si los operadores involucrados en alguna posicion tiene el formato correcto
string getOperadores(string);
```

Para crear el parser se utilizan tres métodos:

```
//Este metodo parsea los tokens generados en expresiones que seran recibidas por el metodo generate
string createLine(int);
```

```
//Metodo que crea las variables a utilizar en el archivo objeto
string createVar(int);
```

```
//Metodo que guarda la referencia a las lineas para realizar los branch (jumps)
string createRefLine(int);
```

Así mediante estos tres métodos se parsean las líneas y se van preparando las líneas de código del archivo fuente para la creación del código intermedio.

```
//Este metodo recibe la informacion previamente analizada y genera el codigo intermedio
void generate(string &);
```

El método que se encarga de insertar el código en el archivo objeto es el método emit(), este método consulta los vectores de caracteres generados por el método generate() y los inserta en el archivo objeto:

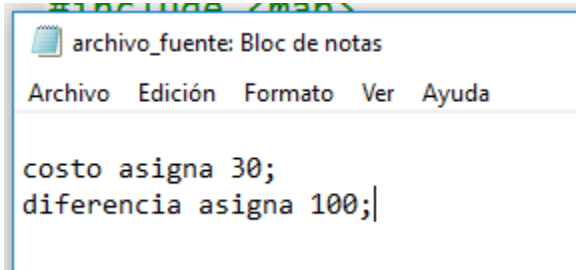
```
//Este metodo se encarga de insertar el codigo en el archivo objeto
void emit();
```

Al momento de terminar de parsear las líneas de código, se deben de limpiar los vectores generados en el momento del parseo, para ello se utiliza el método clearVectors():

```
//Este metodo se utiliza para limpiar los vectores resultantes de la etapa de parseo
void clearVectors();
```

Pruebas

1) Declaración y asignación de variables:



La información proporcionada por el análisis léxico:

costo asigna 30	
identificador	1
numero	2
operador	1
reservada	0
diferencia asigna 100	
identificador	2
numero	5
operador	2
reservada	0

El archivo de salida o archivo fuente muestra el siguiente código:

```
costo=30;
diferencia=100;
```

2) Operaciones de suma, resta, multiplicación y división

Estas operaciones realizan la operación *,/+, - y la asignación:

```
archivo_fuente: Bloc de notas
Archivo Edición Formato Ver Ayuda
costoTotal multiplica(costo,cantidad);
promedio divide(costoTotal,10);
acomulador suma(acomulador,1);
```

Información generada por el análisis léxico:

identificador	3
numero	0
operador	3
reservada	0

La generación de código en el archivo objeto:

```
archivo_objeto: Bloc de notas
Archivo Edición Formato Ver Ayuda
costoTotal=costo*cantidad;
promedio=costoTotal/10;
acomulador=acomulador+1;
```

3) Bucles:

```
archivo_fuente: Bloc de notas
Archivo Edición Formato Ver Ayuda
bucle(i=10,i>0);
precio asigna 34.67;
total suma(precio,precio);
itera(i--,);
```

Genera el siguiente código en el archivo objeto:

```
archivo_objeto: Bloc de notas
Archivo Edición Formato Ver Ayuda
L0:
VAR0=i>0;
if(VAR0)gotoL1;
gotoL2;
L1:
precio=34.67;
total=precio+precio;
i--;
gotoL0;
L2:
```

4) Control de flujo

```
archivo_fuente: Bloc de notas
Archivo Edición Formato Ver Ayuda
edad asigna 17;
condicion(edad>18,);
MayorEdad asigna 1;
fin;
MayorEdad asigna 0;
```

Genera el siguiente código objeto:

```
archivo_objeto: Bloc de notas
Archivo Edición Formato Ver Ayuda
edad=17;
VAR0=edad>18;
if(VAR0)gotoL0;
gotoL1;
L0:
MayorEdad=1;
L1:
MayorEdad=0;
```

5) Funciones

```
archivo_fuente: Bloc de notas
Archivo Edición Formato Ver Ayuda
funcion corre;
corriendo asigna 1;
FIN;
corriendo suma(corriendo,1);
llama corre;
cansado suma(cansado,1);
```

Genera el siguiente código objeto:

```
archivo_objeto: Bloc de notas
Archivo Edición Formato Ver Ayuda
corre:
corriendo=1;
goto L*0:
fin:
corriendo=corriendo+1;
call corre:
L*0:
cansado=cansado+1;
```

Conclusión

El proceso de compilación del presente compilador abarca desde la lectura del archivo fuente hasta la creación del archivo objeto.

Antes de comenzar a crear el compilador se tiene que definir la gramática del lenguaje, sus reglas gramaticales o de producción, el alfabeto y los símbolos terminales y no terminales.

Las líneas de código obtenidas del archivo fuente pasan por varias etapas hasta convertirse en líneas pertenecientes del archivo objeto.

1) Scanner:

Es el encargado de ejecutar un análisis léxico de las líneas del archivo fuente, su proceso consiste en ir obteniendo las líneas de código, las analiza y obtiene información relevante de la gramática del lenguaje impresa en cada una de estas líneas, obtiene los identificadores, palabras reservadas, números enteros y flotantes.

Mediante esta información generada crea los tokens, que son elementos que recibirá la parte del análisis sintáctico y semántico (parser).

Las reglas gramaticales del lenguaje generan patrones para la creación de los tokens.

2) Parsing:

Es el encargado de interpretar los tokens proporcionados por la fase de Scanner, estos son pasados a reglas gramaticales propias del lenguaje, para luego construir una representación de la línea de código proveniente del archivo fuente y de acuerdo a las reglas gramaticales, de esta manera se genera un código o representación del código fuente.

En el proceso de parseo se identifica los elementos del lenguaje a los cuales pertenece cada parte de la línea fuente, se crea una representación de esta línea fuente, sin alterar el funcionamiento o la tarea de esta.

El proceso de análisis semántico se encarga de reconocer las variables o identificadores de variables ingresados, estos id's son insertados en un vector, mediante este análisis se reconocen las palabras reservadas y los identificadores, para posteriormente generar el código intermedio.

En el momento en el que el árbol de parseo identifica tanto las variables asignadas y la acción de cada línea de código comienza la generación de código intermedio.

Este código es insertado en el archivo fuente, el cual ya se tratara de una interpretación del código fuente.