T
001.6424B
Y36

BIBLIOTECA

Escuela Superior Politécnica del Litoral

ESCUELA DE CIENCIAS DE LA COMPUTACION

Basic 1.0

TESIS DE GRADO

Previa a la obtención del TITULO de

ANALISTA DE SISTEMAS

Presentada por:

FERNANDO YCAZA MORLA

ANA MORLA BOLOÑA

Guayaquil - Ecuador

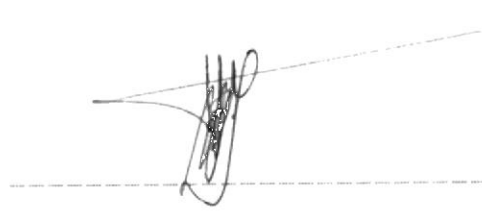
1985

a nuestros padres...

ING. LUIS RODRIGUEZ O.
DIRECTOR DE TESIS

DECLARACION EXPRESA

"La respondabilidad por los hechos, ideas y doctrinas expuestos en esta tesis, corresponden exlusivamente a su autor, y el patrimonio intelectual de la tesis de grado, corresponden a la Escuela Superior Politecnica del Litoral"



FERNANDO YCAZA MORLA

DECLARACION EXPRESA

"La respondabilidad por los hechos, ideas y doctrinas expuestos en esta tesis, corresponden exlusivamente a su autor, y el patrimonio intelectual de la tesis de grado, corresponden a la Escuela Superior Politecnica del Litoral"

ANA MARIA MORLA BOLOÑA

INDICE

1. INTRODUCCION 1

2. OBJETIVOS 2

3. ESTRUCTURA DEL LENGUAJE BASIC 1.0

3.1 Simbolos Utilizados 4

3.2 Variables Utilizadas 5

3.2.1 Identificadores 5

3.2.2 Formato para los numeros 5

3.3 Reglas para evaluacion de expresiones 6

3.4 Estructuras Matematicas 7

3.5 Estructuras Condicionales 8

3.6 Sentencia o Estructura Basica del Lenguaje 9

3.7 Instrucciones de Entrada y Salida 10

3.8 Instrucciones de Transferencia de Control 12

3.9 Instrucciones de Asignacion 14

3.10 Comentarios en el Programa 15

3.11 Palabras Reservadas 16

3.12 Programas Ejemplos 17

4. REQUERIMIENTOS DE IMPLEMENTACION DEL SISTEMA
..... 19

5. OPERACION DEL SISTEMA
..... 20

6. EDITOR
6.1 Entrada de Programas 22
6.2 Modificacion de Programas 26

7. COMPILACION DE PROGRAMAS
7.1 Como utilizar el programa compilador 29
7.2 Posibles errores durante compilacion 31

8. EJECUCION DE PROGRAMAS
8.1 Posibles errores durante compilacion 33

9. CONCEPTOS DE DISENO DE COMPILADORES
Introduccion 35
9.1 Data 36
9.1.1 Organizacion logica, estructura de almacenamiento y su representacion sintactica..... 37
9.1.2 Declaraciones 38
9.1.3 Relaciones entre Datos y Operaciones 39
9.1.4 Items de datos elementales ... 39

9.1.5 Arreglas homogeneos de tamano fijo 40
9.1.6 Arreglos, registros y estructuras heterogeneas de tamano fijo 41
9.1.7 Arreglos de tamano variable 42
9.1.8 Destruccion de estructuras de datos y eliminacion de elementos 47
9.2 SECUENCIA DE CONTROL
9.2.1 Secuencia de control implicita y explicita 48
9.2.2 Sentencias Iterativas 50
9.3 OPERACIONES
..... 52
9.4 SINTAXIS Y TRADUCCION
9.4.1 Elementos sintacticos de un lenguaje 55
9.4.2 Sintaxis para expresiones 57
9.4.3 Etapas en la traduccion 59
9.5 EL LENGUAJE BASIC 1.0
9.5.1 Como opera el BASIC 1.0 62
9.5.2 El programa interprete 66
10. CONCLUSIONES Y RECOMENDACIONES
..... 73
11. BIBLIOGRAFIA
..... 75

1. INTRODUCCION

Todos los lenguajes que conocemos utilizan sus instrucciones en un idioma distinto al de nuestro hablar corriente, lo cual constituye un obstaculo para aquellos que aun sin conocerlo desean aprender "algo" acerca de los conceptos y beneficios de la programacion.

Ofrecemos un lenguaje denominado BASIC 1.0, que tiene similitud de estructuras logicas al Basic, de escritura en espanol aunque con restricciones en el manejo de tipos de variables. Sin embargo servira de base para aquellos que desearen implementar y perfeccionar este lenguaje.

Es tan solo el inicio de una investigacion que desearíamos sea continuada.

2. OBJETIVOS

El uso de un lenguaje de programacion requiere el conocimiento de ciertos conceptos y estructuras logicas que, inicialmente resultan dificiles de entender y poder concretarlos en la practica, por esa razon, nuestro proyecto fue orientado para ademas de palpar muy de cerca la gran complejidad que constituye la construccion de un lenguaje, poder dejar una herramienta para todos aquellos que estan dando los primeros pasos en el mundo de la informatica. Ofrecemos este esfuerzo seguros de que sera una manera de incursionar en el campo de la programacion de computadores.

BIBLIOTECA

3. ESTRUCTURA DEL LENGUAJE BASIC 1.0

En esta seccion se describe la sintaxis del lenguaje BASIC 1.0

3.1 SIMBOLOS UTILIZADOS

- Letras: que se refieren al conjunto que forma nuestro alfabeto.

  A B C D E F ..... W X Y Z

- Digitos: que forman nuestro sistema numerico

  1 2 3 4 5 6 7 8 9 0

- Simbolos Especiales

  Operadores Aritmeticos

  * / + -

  Operadores Relacionales

  = < > >= <= <>

  Otros

  . que representa el punto decimal

  , que servira como elemento de separacion entre identificadores de variables.

- Palabras Reservadas.

  Corresponden a aquellas palabras que serviran para construir las instrucciones del lenguaje.

  Como ejemplo tenemos:

  VAYA SI LUEGO SINO FINSI

  entre otras.

3.2 VARIABLES UTILIZADAS

Una variable hace referencia a una posicion de memoria y en este lenguaje se usara unicamente variables numericas de tipo real (con decimales).

3.2.1 IDENTIFICADORES

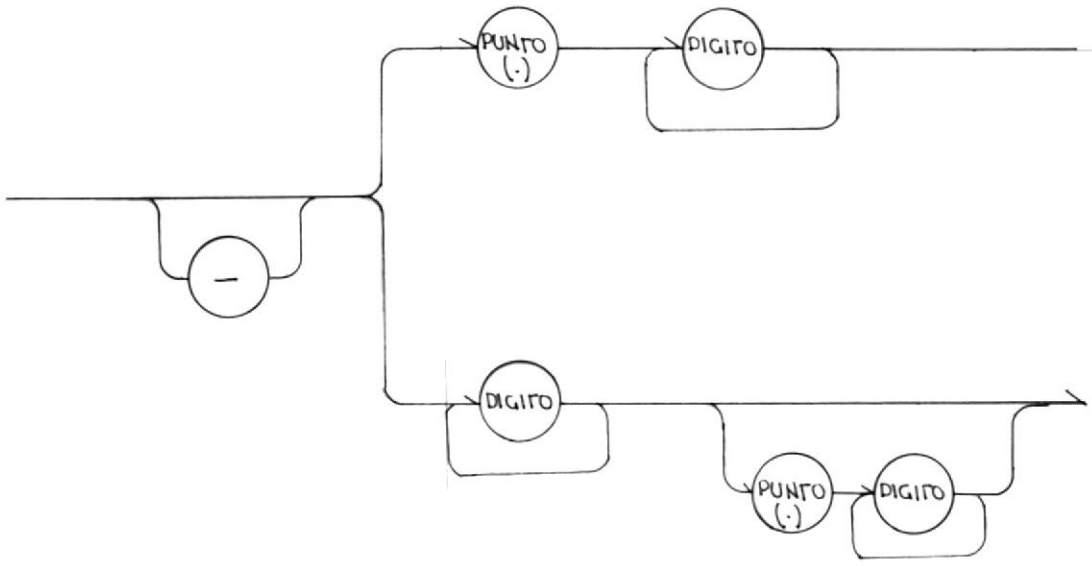
Nombres con el que se reconocen las variables, dichos nombres deberan ser distintos a las palabras reservadas y nunca iguales a los que designan nombres de parrafos.

Su longitud maxima es de 10 posiciones.

Ejemplos: VAR1 VARIABLE

3.2.2 FORMATO PARA LOS NUMEROS

El asignamiento de un numero a una variable, o la utilizacion de un numero como constante se podra hacer bajo el siguiente formato:



Todo numero sera transformado internamente a

formato real y mostrado en formato exponencial o real.

Si el numero de digitos enteros es menor o igual a 6 posiciones el formato en que sera mostrado es real, caso contrario sera mostrado en formato exponencial.

La longitud maxima de los numeros es de 14 posiciones (digitos).

El rango de valores enteros es 32767 a -32768 en cuanto a valores reales es de 32 bits

Asi por ejemplo:

NUMERO	MOSTRADO
6523	6523.00
.987	.99
134567890110	1.34567890110E11

3.3 REGLAS PARA EVALUACION DE EXPRESIONES

La prioridad de los operadores sera definida verticalmente.

( )

* / (igual prioridad, evalua la expresion de izquierda a derecha)

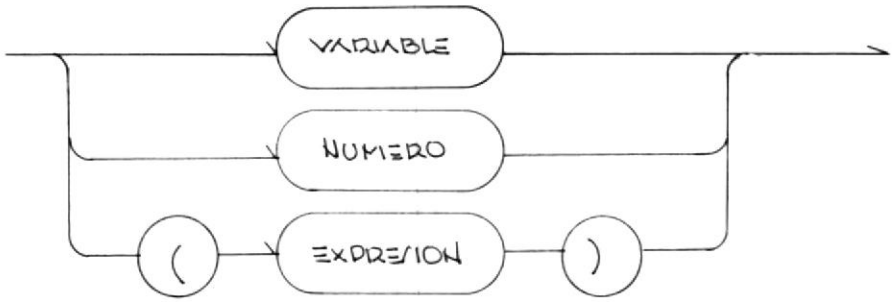
\+ - (igual prioridad, evalua la expresion de izquierda a derecha)

< > >= <= => <> >< = (igual prioridad, evalua la expresion de izquierda a derecha)

3.4 ESTRUCTURAS MATEMATICAS

Las siguientes estructuras seran utilizadas sea en expresiones condicionales o matematicas.

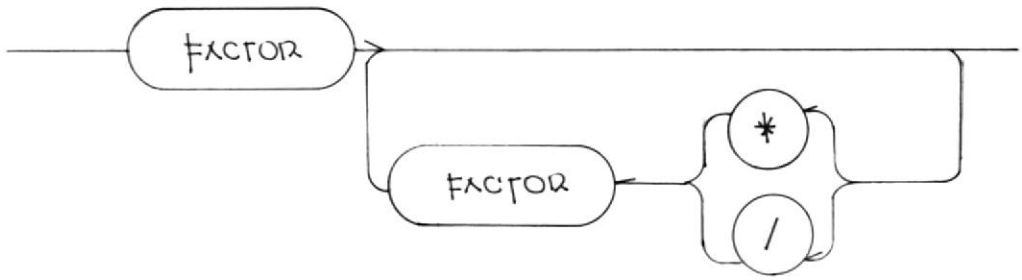
- FACTOR



Ejemplos:

00, 28765, (A+B*X+Y/3)

- TERMINO

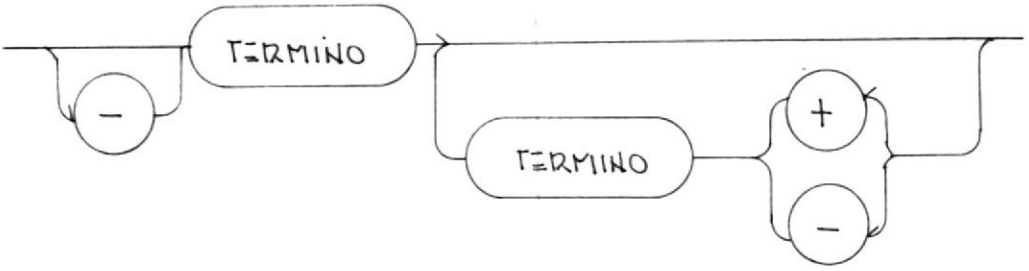


Ejemplos:

(A+B) * 3598

(B-5-ZZ) * H

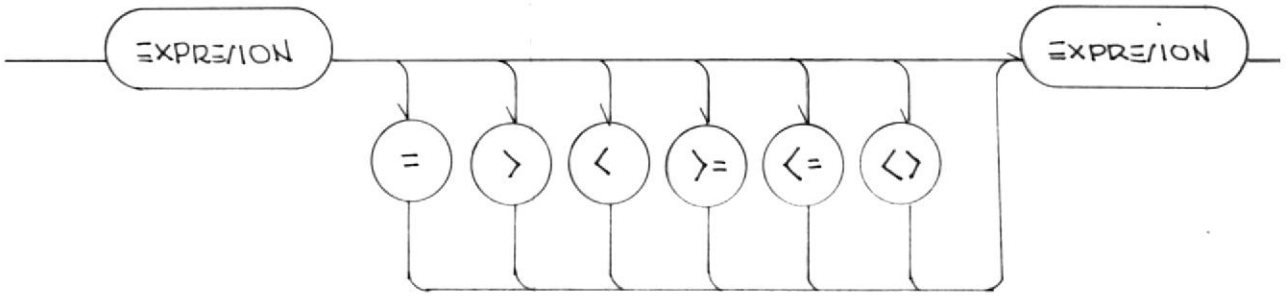
- EXPRESION MATEMATICA



Ejemplos:

W + Z , -230+(Q+R) + Y

3.5 ESTRUCTURAS CONDICIONALES

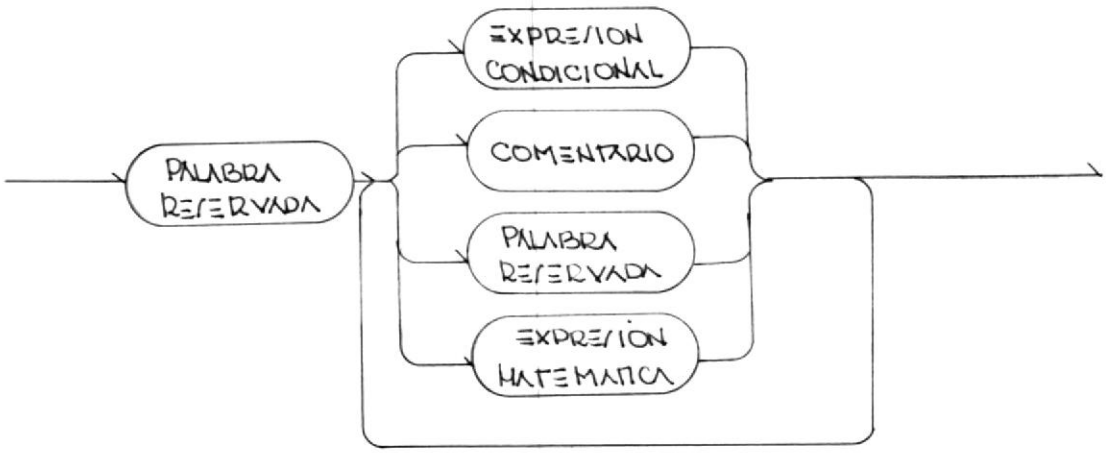


Ejemplo:

A > 3, A < (3+W)

3.6 SENTENCIA O ESTRUCTURA BASICA DEL LENGUAJE

Toda sentencia del lenguaje debera guardar la siguiente sintaxis:



Podra ser escrita en una o varias lineas.

Las lineas en blanco seran ignoradas.

Al conjunto de sentencias se las reconocera con el nombre de BLOQUE, a este bloque se le podra o no dar el nombre mediante una instruccion (PARRAFO), todo ello para fines de estructurar logicamente el programa.

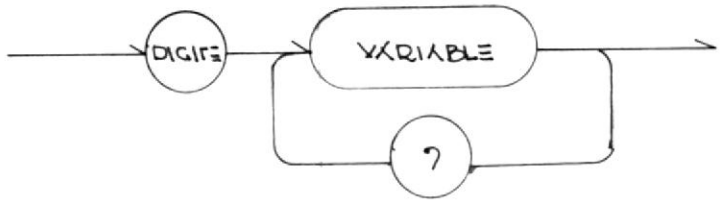
El conjunto de bloques constituira el PROGRAMA.

Podemos decir entonces que programa es el conjunto de sentencias del lenguaje codificadas de manera logica tal que cumplan el objetivo que impulso a escribirlas.

3.7 INSTRUCCIONES DE ENTRADA Y SALIDA

DIGITE

Sintaxis.



Dicha instruccion hara referencia a la pantalla y para saber el momento en que se debera escribir los valores con que deseamos cargar la o las variables especificadas en la sentencia aparecera un mensaje:

INGRESE DATOS

Como sabemos el tipo de variables aceptadas por el lenguaje solo son numericas sean enteras o reales.
Cada dato estara separado uno de otro mediante blancos asi por ejemplo si pedimos:

(en el programa) DIGITE A, B

(durante ejecucion) INGRESE DATOS

solo en este momento podremos ingresar los valores que deseemos.
Los valores seran validados de acuerdo a las reglas que se refieren al formato de los numeros, y luego asignados a las variables respectivamente.

Si en algun momento durante la entrada de datos, no respetamos lo especificado en el FORMATO DE NUMEROS

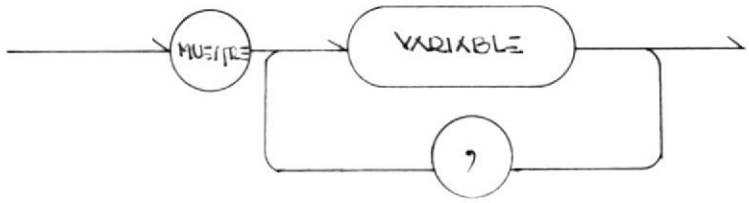
tendremos el error durante ejecucion:

ERROR EN DATOS

y automaticamente abandonaremos el programa.

MUESTRE

Sintaxis.



Otra instruccion orientada a la pantalla, mostrando a traves de ella el valor de cada una de las variables especificadas en la instruccion.

Cada valor sera mostrado uno al lado del otro, ocupando 16 posiciones de la pantalla cada vez.

Los numeros seran mostrados con minimo dos posiciones decimales, a pesar de que hubieran sido enteros al ingresarlos como constantes o asignados a variables.

Ejemplos:

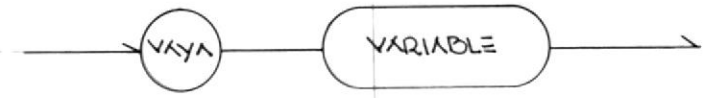
(en el programa) MUESTRE A,B

(durante ejecucion) 1025.00 2040.00

3.8 INSTRUCCIONES DE TRANSFERENCIA DE CONTROL

VAYA

Sintaxis.



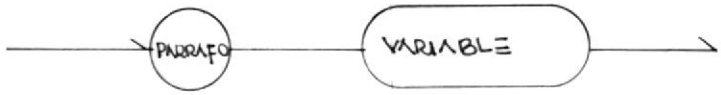
Esta instruccion no es otra cosa que un salto incondicional al identificador que hace referencia a un bloque.

Ejemplo:

VAYA PARAFITO1

PARRAFO

Sintaxis.

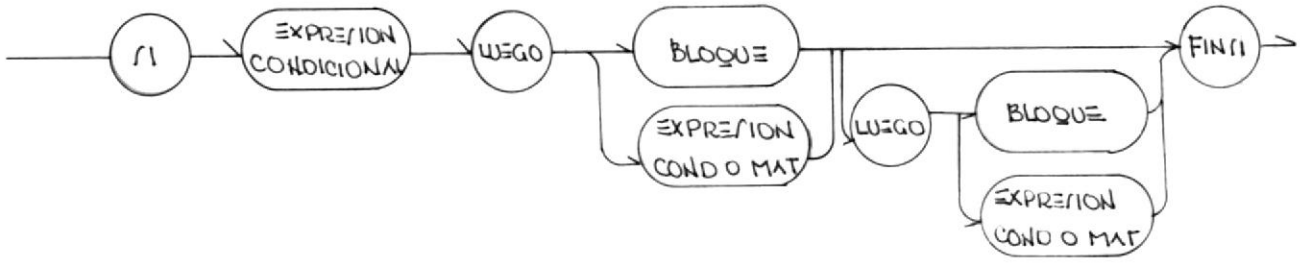


Servira para estructurar logicamente en bloques nuestro programa.

Esta es la instruccion "destino" a la que la instruccion VAYA incondicionalmente transmitira el control.

SI LUEGO SINO FINSI

Sintaxis.



Mediante esta instruccion podemos escoger dos caminos, y el escogido dependera unicamente del resultado de la evaluacion de la expresion condicional.

Si dicho resultado es verdadero se tomara el camino de las instruciones que siguen a la palabra LUEGO.

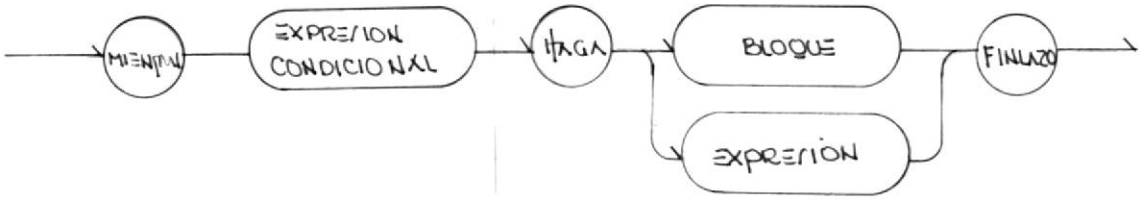
Una vez ejecutadas estas sentencias, el control pasara a la siguiente instruccion luego de la palabra FINSI.

En el caso de que la condicion hubiera resultado falsa, debera ejecutar las sentencias que siguen a la palabra SINO o si esta no hubiera sido especificada, el control pasaria a la siguiente instruccion que sigue a FINSI.

```
SI A > B   LUEGO
   ASIGNE  A = 50
   DIGITE C
   SI C < 3
      LUEGO
      B = 8
   FINSI
   SINO
   A = 8
   FINSI
```

MIENTRAS HAGA FINLAZO

Sintaxis



Permite ejecutar un grupo de instrucciones mientras se cumpla una condicion.

Se evaluara la condicion, segun ello se ejecutaran o no la serie de instrucciones que estan dentro del lazo.

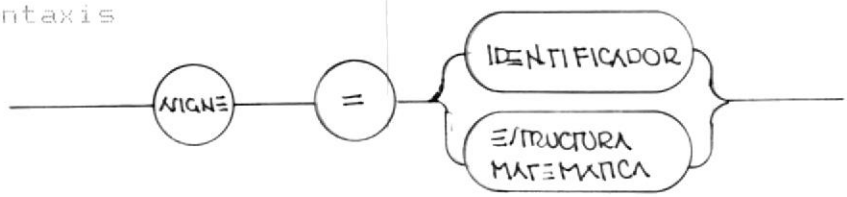
Luego de la ultima instruccion el control pasara a la primera instruccion dentro del lazo, procediendo nuevamente a evaluar la condicion, segun ello volvera o no a ejecutar la instruccion.

```
MIENTRAS A > 8
  HAGA
    ASIGNE A = A + 3
FINLAZO
```

3.9 INSTRUCCIONES DE ASIGNACION

ASIGNE

Sintaxis



Para darle el valor a una variable ya sea mediante un numero o dandole el mismo valor que otra variable o tal

vez una expresion matematica mas compleja.

Ejemplo:

ASIGNE A = 56 , ASIGNE B = (U+78)

3.10 COMENTARIOS EN EL PROGRAMA

COM

Servira de ayuda para documentar nuestro programa. Cada vez que se encuentre esta instruccion, todo caracter hasta el fin de la linea sera considerado como comentario.

3.11. PALABRAS RESERVADAS, VARIABLES

Las siguientes palabras no deben usarse como nombres de variables en el programa.

DIGITE

MUESTRE

VAYA

PARRAFO

SI LUEGO SINO FINSI

MIENTRAS HAGA FINLAZO

COM

ASIGNE

En cuanto a variables, en esta version solo se permite de tipo real (internamente) como ya se lo explico.

3.12 PROGRAMAS EJEMPLOS

```
COM    PROGRAMA QUE CALCULA EL FACTORIAL DE UN
COM    NUMERO
COM
COM    PIDE NUMERO A CALCULAR POR PANTALLA
COM    SI EL DATO A CALCULAR ES -1 INDICA FIN DE
COM    PROGRAMA


       PARRAFO PIDEDATO
       DIGITE N
       SI N = -1
                  LUEGO VAYA FINAL
       FINSI

COM    PROCESO DE CALCULOS

       ASIGNE PARCIAL = N
       PARRAFO PROCESO
       SI N <= 1
                  LUEGO VAYA TOTAL
       FINSI
       ASIGNE N = N - 1
       VAYA PROCESO

COM    MOSTRAR EL RESULTADO POR PANTALLA

       PARRAFO TOTAL
       MUESTRE PARCIAL
       VAYA PIDEDATO

       PARRAFO FINAL
       FIN


COM    PROGRAMA QUE CALCULA LOS DIVISORES DE
COM    UN NUMERO
COM
COM    PIDE NUMERO POR PANTALLA Y SI ESTE ES
COM    IGUAL A -1 INDICA FIN DE PROGRAMA

       PARRAFO PIDEDATO
       DIGITE N
       SI N = -1
                  LUEGO VAYA FINAL

       FINSI

COM    PROCESO DE CALCULOS

       ASIGNE PARCIAL = N
```

```
      PARRAFO PROCESO
      SI PARCIAL < 1
                        LUEGO VAYA PIDEDATO
      FINSI
      ASIGNE W = N / PARCIAL
      SI W1 <> 0
                     LUEGO MUESTRE PARCIAL
      FINSI
      ASIGNE PARCIAL = PARCIAL - 1
      VAYA PROCESO

      PARRAFO FINAL
      FIN
```

```
COM
COM     PROGRAMA EJEMPLO QUE COMBINA LAS DISTIN-
COM     TAS SENTENCIAS Y OPERACIONES DISPONI   -
COM     BLES EN ESTE LENGUAJE BASIC 1.0
```

```
 DIGITE A,B
 MIENTRAS A < B
           HAGA
                ASIGNE C= 8*B+A
                MUESTRE C
           FINLAZO
 SI A > C
          LUEGO
               ASIGNE D = .303+C
          SINO
               VAYA BLOQUE
          FINSI
 SI A > 8000
         LUEGO MIENTRAS  X > 20
               HAGA
                    ASIGNE X = X + 1
                    MUESTRE X
               FINLAZO
         FINSI
 PARRAFO BLOQUE
 MUESTRE D,C
 ASIGNE Y = (X*4+A+D/C)
 ASIGNE Z = A + B + C +D
```

4. REQUERIMIENTOS DE IMPLEMENTACION DEL SISTEMA

La implementacion de este sistema es en un APPLE II e.

El sistema requiere basicamente:

- Memoria de por lo menos 64 bytes.
- Dos disk drives.
- Dos diskettes que contienen programas y el sistema operativo pascal.
- Manual operativo del Lenguaje.

BIBLIOTECA

5. OPERACION DEL SISTEMA

1.- Inserte diskettes en drives correspondientes segun el numero que tienen especificado.

2.- Encienda el computador. (APPLE II e)

3.- Los programas seran creados utilizando el editor de Pascal.

4.- Luego podra proceder a compilar y ejecutar el programa.

6. EDITOR

6.1 ENTRADA DE PROGRAMAS

Al hablar de escribir un programa nos estamos refiriendo a la accion de editar la codificacion del programa o correcciones a programas ya ingresados. Para esto es necesario hacer uso del editor del Pascal.

Despues de seguir los pasos de operacion del sistema se presentaran las opciones:

```
Command(F(ile   E(dit   C(ompile   A(ssemble   L(ink
```

De estas opciones usaremos E que se refiere al Editor.

En este momento con solo presionar **RETURN** podremos comenzar a digitar nuestro programa.

Es necesario hacer una breve explicacion de la linea de comandos que podemos manejar dentro del Editor, para ello tomaremos en cuenta unicamente los que realmente nos seran de utilidad.

Se nos presenta la linea de comandos:

```
EDIT: A(DJST C(PY D(LETE F(IND I(NSRT ...Q(UIT...
```

A partir de esta linea de comandos podremos escoger cualquier opcion, tan solo digitando la primera letra:

INSERTAR I(NSRT

Para insertar cualquier letra, numero o caracter debera primero mover el cursor a la posicion correcta y entonces tipear la "I" . Recuerde, siempre debera mover

el cursor a la posicion correcta antes de tipear la "I". Para tener la seguridad de que se esta en el modo de INSERT debera encontrarse en la cabecera de la pantalla la siguiente linea

>INSERT: TEXT [<BS> A CHAR, <DEL> A LINE].....

Si nosotros hemos escrito la palabra MDELO y lo que quisimos poner era MODELO deberemos entonces posicionar el cursor en la letra D y en este momento pedir el modo INSERT (--> I) y podremos entonces colocar la letra O que es la que nos hacia falta.
Y para salir del modo INSERT unicamente sera necesario presionar la tecla <Control> y la tecla C, ambas al mismo tiempo.

DELETE D(LETE

Nos sera de utilidad en el momento en que quisieremos eliminar ya sea un caracter o lineas enteras que estuvieren demas.
Esta vez como indicativo de estar en el modo DELETE tendremos la linea cabecera asi:

>DELETE: <> <MOVING COMMANDS> [<ETX>TO DELETE, <ESC> TO ABORT]

recordando que EXT no es otra cosa que <Control> y la tecla C al mismo tiempo.
Asi tambien como en el INSERT deberemos posicionarnos, esta vez en el caracter o en el comienzo de linea que

quisieramos eliminar, presionamos la **D** con la que activamos este modo y si nuestro caso fuera eliminar caracteres unicamente deberemos presionar la tecla de la flecha a la derecha con la que iremos eliminando caracter por caracter, si nuestro deseo fuera eliminar una linea entera podremos presionar <RETURN>.

Una vez eliminados todos los caracteres o lineas innecesarios a nuestros requerimientos, presionaremos <Control> y la tecla **C** .

Como detalle adicional podemos decir que si antes de dar <Control> **C**, presionamos la tecla de la flecha a la izquierda observaremos que los caracteres que habian desaparecido, vuelven ha estar en la pantalla, hablamos de una manera de reversar lo borrado en caso de que nos equivocaramos.

QUIT Q(UIT

Cuando todas las sentencias esten escritas y el programa acabado, sera momento de grabarlo, para esto deberemos presionar la tecla **Q** y se nos presentaran una serie de opciones:

U(PDATE THE WORKFILE AND LEAVE

E(XIT WITHOUT UPDATING

R(ETURN TO THE EDITOR WITHOUT UPDATING

W(RITE TO A FILE AND RETURN

S(AVE WITH SAME NAME AND RETURN

De todas estas opciones sera conveniente escoger la

ultima **S** podremos en este momento dar el nombre a nuestro programa, con este nombre lo reconoceremos para que sea luego compilado y ejecutado.

Si no se deseara grabar el programa o las actualizaciones que se hubieren hecho se debera tomar la opcion **E** de Exit o Salida.

Si deseara regresar al editor antes de grabar, escoger la opcion **R**.

Si el programa ya ha sido grabado antes, para grabar las actualizaciones hechas, escoger la opcion **S** y aparecera una opcion a la que podremos contestar con **Y** de manera que borraremos la version anterior del programa, y claro quedando la ultima actualizacion.

El nombre del programa debera ser especificado de la siguiente manera:

#4:NOMBRE.TEXT

(constantes)
(de hasta 6 caracteres)
(unidad utilizada puede ser 4 o 5)

Ejemplo:

#4:PRUEBA.TEXT

Si nosotros escogemos el drive 4 para grabar nuestros programas no sera necesario especificar el numero de drive ni escribir los dos puntos (:).

No recomendamos grabar muchos programas en el drive 4 por que es ahi donde se encuentran los programas que hacen posible este lenguaje y ocupan gran cantidad de la capacidad de los diskettes.

COMANDOS DE MOVIMIENTOS

Muchas veces es necesario en el momento en que se esta digitando el programa, desplazarnos a otra posicion dentro del fuente del programa, mas alla de las lineas que se muestran en la pantalla, para ello podemos hacerlo de dos maneras:

- presionando la tecla **P**, con la que avanzaremos toda una pantalla.
- presionando **<Control>** y la tecla **O** con la que avanzaremos una linea.

Cabe recalcar que si es nuestro deseo invertir el sentido en que se mostraria el programa, esto es: en vez de subirlo, bajarlo, lo unico que tenemos que alterar es el sentido del signo (< >) que se encuentra delante de la palabra **EDIT** en la linea que encabeza nuestro programa o linea guia del Editor.

6.2 MODIFICACION DE PROGRAMAS

Al igual que para crear un programa, tomaremos la opcion E(DIT presionando como antes la tecla **E**, la diferencia sera que ahora si tendremos que poner al principio el nombre de aquel programa que deseamos corregir.

El nombre sera igual al que especificamos en el momento que grabamos el programa.

Por todo lo demas, es decir los comandos usados, la actualizacion en todo momento funcionara de la misma manera que en el momento que fue ingresado por primera

vez el programa.

7. COMPILACION DE PROGRAMAS

7.1 COMO UTILIZAR EL PROGRAMA COMPILADOR

Nuestro programa previamente digitado no es todavia apto para que se ejecute, se hace necesario transformarlo a un lenguaje que sea mas facil de entender para la maquina, este proceso de traducir el programa digitado a otro que sera ejecutado es lo que se llama COMPILACION.

Para grabar nuestro programa fuente utilizamos el EDITOR PASCAL opcion a la que se llegaba digitando la E dentro de la linea de comandos:

```
Command(f(ile   E(dit   C(ompile   A(ssemble   L(ink
X(cute
```

Ahora deberemos utilizar la opcion X, inmediatamente pedira nombre del programa a ejecutar, a ello deberemos responder :

```
#5:COMPIL.CODE
```

debe ser escrito en forma identica a la que aqui expresamos de lo contrario no se ejecutara ninguna compilacion.

Una vez que el programa se esta ejecutando el requerimiento unico sera conocer el nombre del programa que nosotros deseamos compilar, este se refiere al que nosotros escribimos.

Otra pregunta adicional que debera ser contestada se refiere a la opcion de listar la compilacion por impresora aunque siempre se listara por la pantalla.

Al listar linea por linea la compilacion, nos dara una

alerta en el momento en que encuentra un error, permitiendonos terminar si fuera ese nuestro deseo para lo que digitaremos <Esc>, o <Return> si queremos continuar.

7.2 POSIBLES ERRORES DURANTE COMPILACION

1) Numero muy grande
2) Numero incorrecto
3) Parentesis no balanceados
4) Campo debe ser identificador
5) Nombre de procedimiento ya existe
6) Identificador previamente usado como etiqueta
7) Simbolo no reconocido dentro de un factor
8) Operador relacional debe seguir a la condicion
9) Identificador debe ser seguido por '='
10) Sentencia debe ser seguida por un identificador
11) 'LUEGO' no especificado en sentencia 'SI'
12) 'SINO' no empareja con un 'SI'
13) 'FINSI' no empareja con un 'SI'
14) 'HAGA' no especificado en sentencia 'MIENTRAS'
15) 'FINLAZO' no empareja con un 'MIENTRAS'
16) Sentencia no reconocida
17) Fin de estructuras de control no especificadas
18) 'FIN' de programa no especificado
19) Etiqueta de bifurcacion no definida
20) Excesiva cantidad de Parrafos
21) Excesiva cantidad de identificadores (campos)
22) Nombre de Parrafo ha sido usado antes como campo

8. EJECUCION DE PROGRAMAS

Corresponde a la ejecucion de las instrucciones que conforman el programa.

Para llegar al momento de la ejecucion, deberemos siempre, como se indico en el capitulo anterior, empezar compilando el programa , para asi obtener de este proceso el seudo-assembler que sera el que se ejecute finalmente.

Si el programa hubiera culminado la compilacion exitosamente se mostrara la opcion:

Desea ejecutar el programa (S/N)

Podremos entonces tomar la decision de ejecutar el programa e inmediatamente comenzara.

Como opcion adicional durante la ejecucion del programa tenemos la de listar o no el codigo objeto, que no es otra cosa que el equivalente al asemmbler de cualquier otro lenguaje, en nuestro lenguaje.

8.1 POSIBLES ERRORES DURANTE EJECUCION

- Division para 0 no permitida.

  Si en algun momento se presenta el caso de la division para (0) sera mostrado dicho mensaje de error.

9. CONCEPTOS DE DISEÑO DE COMPILADORES

CARACTERISTICAS DE UN BUEN LENGUAJE

El diseno de un buen lenguaje de programacion requiere ciertas caracteristicas. El programador debe conocer las debilidades y fortalezas de un lenguaje. Que cosas el debe buscar en un lenguaje ?

1) Claridad, Simplicidad y Unidad en la concepcion del lenguaje.- Un lenguaje de programacion debe proveer una contextura conceptual para pensar acerca de algoritmos y medios de expresar estos algoritmos para la ejecucion en la maquina. Deberia proveer al programador en forma clara, simple un set de instrucciones que pueden usarse como primitivos en el desarrollo de algoritmos.

2) Claridad en la Estructura del Programa.- Mas importante es para el programador un lenguaje que permita 'estructurar' sus instrucciones para reflejar claramente el algoritmo usado. Los beneficios de un programa bien estructurado se reflejan al escribir, rastrear, modificar y entender el programa. Un buen diseno puede ser malogrado por una mala sintaxis lo cual hace dificil la tarea de programar.

3) Sencillez para su aplicacion.- Esto se logra definiendo apropiadas estructuras de datos, operaciones, estructuras de control y una

sintaxis mas natural para el programa que va a ser resuelto.

Las partes principales en las que un compilador se constituye son las siguientes : Data, Operaciones, Secuencia de control, Control de datos, Manejo de almacenamiento, Medio ambiente, Sintaxis y Traduccion. De estas solo mencionaremos las principales.

9.1 D A T A

Los datos que existen durante la ejecucion de un programa se encuadran en dos categorias : datos definidos por el programador y datos definidos que son reservados por el sistema.

Los datos definidos por el programador hacen referencia a items de datos, por ejemplo : numeros, arreglos, archivos de entrada/salida, que el programador define y manipula dentro de su programa.

Los datos definidos y reservados por el sistema son aquellos items de datos en que la maquina guarda y reserva durante su ejecucion, por ejemplo : stacks que manejan apuntadores de subprogramas, descriptores de estructuras de datos, lista de espacios libres, I/O buffers, etc. Son generados automaticamente por el sistema y no necesitan definicion explicita por el programador.

9.1.1 ORGANIZACION LOGICA, ESTRUCTURA DE ALMACENAMIENTO Y SU REPRESENTACION SINTACTICA

Cuando un programador usa un lenguage para resolver un problema, el debe primero decidir sobre la forma de representar o codificar los datos en terminos de estructuras de datos provistas por el lenguage.

De igual manera cuando un lenguage es implementado sobre un determinado computador se debe de considerar el tiempo de ejecucion y el almacenamiento de datos a ocupar.

Las memorias en general son estructuradas como simples secuencias de bits, divididas en intervalos fijos de 'palabras' o 'bytes' direccionables. Por ejemplo un numero puede ser representado por una secuencia de bits, un apuntador por una secuencia mas corta y un arreglo de numeros por una secuencia mas larga.

Para representar un almacenamiento para un dato dado se considerara :

1) una localizacion de memoria (direccion)
2) una cadena de bits representando los datos codificados, y
3) un descriptor especificando la information adicional necesaria para decodificar la cadena de bits.

Un descriptor incluye un selector de tipos de

datos que especifica la clase de items a la que pertenece.

La cadena de bits encodifica un item que puede existir en un bloque contiguo de memoria.

En este caso nosotros hablamos de datos como almacenamientos secuenciales.

Alternativamente una cadena puede ser dividida dentro de un numero de bloques no contiguos de memoria y encadenadas por apuntadores. En este caso se dice que los datos son encadenados.

Secuencial y encadenamiento son las tecnicas mas usadas para almacenamiento.

9.1.2 DECLARACIONES

Una declaracion es una sentencia que sirve para comunicar al traductor del lenguage, informacion acerca de las propiedades de los datos durante la ejecucion, por ejemplo :

A : ARRAY OF 1..20 OF INTEGER

esta declaracion especifica que un arreglo va a ser creado, su tipo de datos, el numero de sus elementos, los subscritos que van a ser usados para accesarlos, el tipo de dato por cada elemento y el nombre por el cual va a ser referenciado.

Las ventajas de las declaraciones en los lenguages son:

1) almacenamiento y acceso mas eficiente a sus datos

2) mejor manejo de almacenamiento

3) chequeo de tipos de datos ya definidos

9.1.3 RELACIONES ENTRE DATOS Y OPERACIONES

Nos referiremos a la operacion de hacer disponibles elementos de una estructura dada. Por ejemplo los metodos de acceso a elementos de un arreglo siempre necesariamente deben ser discutidos al mismo tiempo que el almacenamiento de un arreglo.

Ciertos tipos de datos pueden ser accesado solamente como un todo, por ejemplo numeros,valores booleanos y apuntadores. Son llamados informalmente items de datos elementales. Con otros tipos de datos, accesar a sub-partes de datos es permitido. Estos son llamados items de datos estructurados.

9.1.4 ITEMS DE DATOS ELEMENTALES

Un item elemental es aquel que puede ser accesado y modificado como una unidad. Numeros, caracteres, strings, valores booleanos, cadenas de bits, simbolo de apuntadores son lo basicos considerados.

Numeros: Las varias clases de numeros son relativamente familiares y sus detalles varian de lenguaje a lenguaje.

Caracteres y Cadenas de Strings: Se define a una cadena de caracteres como una secuencia individual

de caracteres.

Valores Booleanos: La mayoria de los lenguajes soportan alguna forma de datos binarios (verdadero o falso). El bit es llamado dato o valor booleano.

Apuntadores: Tambien se los conoce como referencia, localizacion, direccion.

Ordinariamente los apuntadores no estan directamente disponibles como un campo de usuario, pero puede ser usado como modelo para almacenar otros items.

9.1.5 ARREGLOS HOMOGENEOS DE TAMANO FIJO

Hacemos referencia a la mayoria de estructuras de datos encontrados en un lenguaje de programacion: vectores, arreglos, listas, stacks, colas, registros, arboles, etc. A pesar de la variedad, unas pocas son las principales.

Arreglos lineales pueden ser homogeneos (elementos con el mismo descriptor) o heterogeneos (elementos con varios descriptores). Los descriptores pueden ser especificados cuando el programa sea escrito (declaracion de descriptores) o dejados sin especificar hasta la ejecucion (descriptores a tiempo de ejecucion); y el numero de elementos en el arreglo debe ser fijo a lo largo de la ejecucion (tamano fijo) o puede variar dinamicamente durante la insercion o eliminacion de elementos (tamano variable).

9.1.6 ARREGLOS, REGISTROS Y ESTRUCTURAS HETEROGENEAS DE TAMANO FIJO

Arreglos Lineales heterogeneos con declaraciones.

Los registros de Cobol y las estructuras de PL/I son las mas comunes. Tenemos el siguiente ejemplo en PL/I:

```
DECLARE   1  EMPLEADO,
             2  NOMBRE CHARACTER(15),
             2  EDAD   FIXED,
             2  SALARIO FLOAT;
```

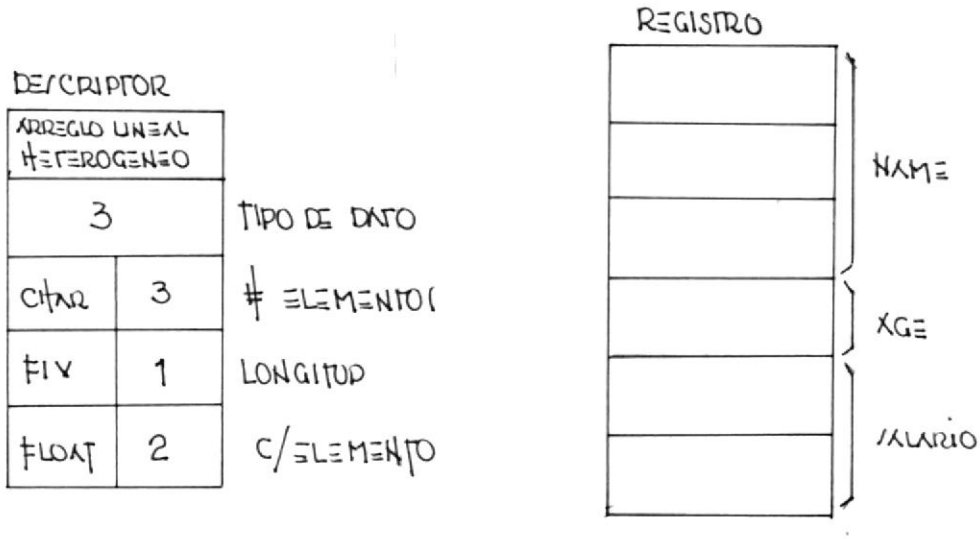
esta declaracion define un arreglo lineal heterogeneo de tres elementos. El primero es una cadena de caracteres, el segundo es un entero y el tercero es un numero real.

Estructura de Almacenamiento.

Se usa una representacion secuencial, con la cadena de bits representando elementos individuales almacenados secuencialmente en un bloque de memoria. El descriptor para este arreglo debe ser mas complejo que para un vector por la variacion de sus elementos. Un descriptor completo consistiria de

1) Tipo de dato de arreglo lineal heterogeneo
2) El numero de elementos
3) Un descriptor para cada elemento.

Ejemplo:



Modo de Acceso.

Su modo de acceso es diferente que la de un vector. En un vector los suscritos enteros sirven para un doble proposito: ellos no solamente permiten elementos individuales del vector para ser accesados al azar, sino que es conveniente procesar el vector entero secuencialmente, usando una variable entera como apuntador.

La variable es inicializada para designar el primer elemento del vector y luego es incrementada para apuntar a cada elemento del vector. Esta clasificacion producto de un proceso secuencial es importante cuando trabaja con vectores.

9.1.7 ARREGLOS DE TAMANO VARIABLE

Estos datos pueden ser ingresados por medio de un dispositivo de entrada y su direccionamiento puede ser generado en forma interna dentro del

programa. Estructuras de tamano variable se los conoce de diferentes maneras: stacks, colas, conjuntos (sets), listas, tablas, para nombrar unos pocos.

Estas estructuras se caracterizan por la manera como crecen y se reducen. Cada elemento debera ser individualmente accesado por subscritos; su acceso tiende a ser relativo.

STACKS

Un stack es una lista cuya modalidad de manejo es la tecnica LIFO(last-in-first-out). El ultimo que entra es el primero que sale. La insercion y eliminacion de elementos es limitado al extremo del stack, llamado tope (top).

Estructura de almacenamiento.

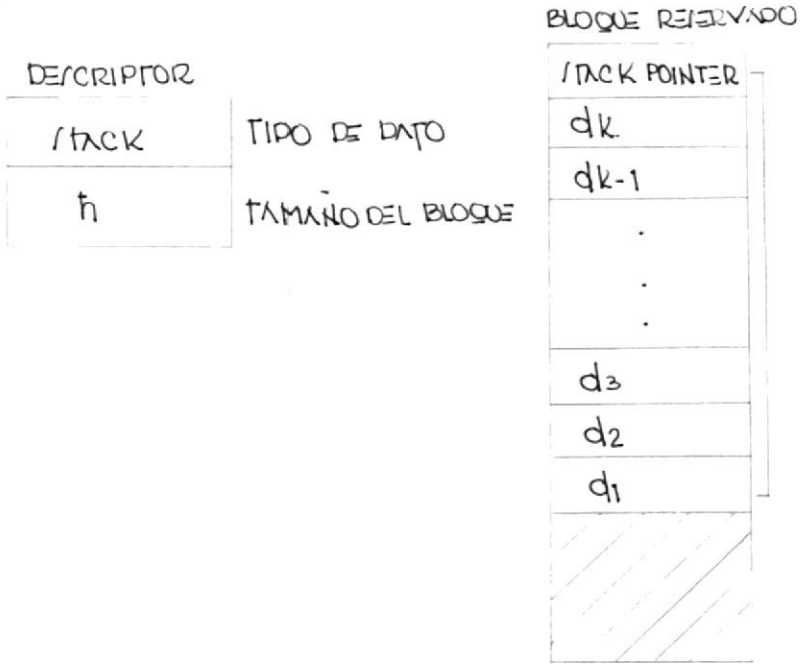
Usa dos modalidades de almacenamiento : la una que es secuencial y la otra es enlazada (linked).

La enlazada es la mas aplicable generalmente para un arbitrario numero de stacks. En caso contrario asumiremos que los elementos del stack son homogeneos o han sido reemplazados por apuntadores a elementos actuales de modo que ellos aparecen homogeneos.

El stack secuencial requiere que un bloque de memoria sea reservado con espacio suficiente como para almacenar el crecimiento del stack a su maxima capacidad.

El primer elemento del bloque es el apuntador a tope del stack. Cuando este apuntador apunta a si mismo, se dice que el stack esta vacio, cuando los elementos son adicionados al stack, ellos son colocados en direcciones consecutivas dentro del bloque reservado y el apuntador al stack es actualizado convenientemente.

Cuando un elemento es eliminado, este debe ser sacado del tope del stack y decrementa el apuntador del stack.



COLAS

Colas usan la tecnica FIFO(first-in-first-out), el primero que entra es el primero que sale.

Esta es distinta que la de los stacks solamente en la insercion de elementos nuevos al fondo de la lista. Las colas crecen en el fondo y se eliminan por el tope.

Estructuras de almacenamiento.

Tanto la forma secuencial como la enlazada son usadas en colas. Se usan dos apuntadores para accesar el stack: el uno que apunta al tope del stack y el otro al fondo del stack.

La cola crece en forma circular en el bloque de esta manera cuando se detecta el fin de la cola esta se incrementa en forma circular, o sea, al comienzo del bloque.

Cuando se elimina un elemento ocasiona que los apuntadores del tope y del fondo coincidan en el caso de que quede vacia

LISTAS

Es un arreglo lineal de tamano variable en el cual las inserciones y eliminaciones son hechas a arbitrarios apuntadores. Acceso de elementos en una lista es usualmente restringida al primer elemento y su seguimiento correspondiente.

Estructura de almacenamiento.

Listas encadenadas y listas simples tienen sus tecnicas semejantes:

Listas simples: cada elemento en la lista es enlazado a su sucesor en la lista, y su apuntador al primer elemento se mantiene(elemento-cabeza de la lista).

Listas doblemente enlazadas: cada elemento de la lista es enlazado tanto a su elemento como a su

predecesor, ambos apuntadores son mantenidos (actualizados) en el elemento cabeza.

CONJUNTOS

Existen 3 operaciones basicas:

1) pertenencia del conjunto: cuando un elemento dado se encuentra en el conjunto.

2) adicion de un elemento: ingreso de un elemento a un conjunto.

3) eliminacion de un elemento: extraccion de un elemento del conjunto.

Estructura de almacenamiento

Puede ser representado como un arreglo lineal de tamano variable, cuando una modalidad de almacenamiento sugeridos para stacks, colas, o listas enlazadas.

Las tecnicas principales son:

1) descomposicion del conjunto: si se modifica el conjunto, el resultado genera una nueva direccion.

2) busqueda secuencial: desde el punto original de colision se procede a buscar en forma secuencial hasta encontrar el elemento, o si no esta, se genera una nueva direccion.

3) bucket: en vez de colocar en forma directa el dato, lo podemos sustituir en apuntadores enlazados al conjunto teniendo la misma direccion.

9.1.8 DESTRUCCION DE ESTRUCTURAS DE DATOS Y ELIMINACION DE ELEMENTOS

La estructura es destruida cuando todos los caminos a la estructura han sido destruidos, al mismo tiempo que las direcciones de almancenamiento pueden ser reusadas.

Cuando una estructura de datos es creada, un camino de acceso a la estructura debe ser creado. La manera usual de hacerlo es nombrandolo por medio de un identificador asociado o almacenar un apuntador dentro de otra estructura. Durante el tiempo de vida de la estructura el camino de acceso creado puede ser aumentado por otro camino de acceso. Por lo tanto habria multiples caminos de acceso a una unica estructura. Esta variedad de caminos a una sola estructura es el principal problema de la eliminacion.

Si se va a destruir una estructura en especial por un camino de acceso se destruiria la estructura quedando los otros caminos de acceso sin su estructura correspondiente.

Referencias "flotantes" son ocasionadas por la recuperacion de almacenamiento para una estructura destruida antes que todos los caminos de acceso hayan sido destrozados. El problema complementario es el "desecho". Referencias flotantes pueden ser evitadas usando una destruccion que elimina un camino de acceso sin

retornar la asignacion de almacenamiento para reusarla hasta que el ultimo camino de acceso haya sido eliminado.

9.2 SECUENCIA DE CONTROL

Estructuras de control en un lenguaje de programacion involucran dos aspectos : el control del orden de ejecucion de las operaciones y la transmision de datos entre conjuntos de operaciones llamados control de datos.

9.2.1 SECUENCIA DE CONTROL IMPLICITA Y EXPLICITA

Diferentes estructuras de control son usadas en varios lenguajes. Ellos pueden ser convenientemente caracterizados en tres grupos :

1) Estructuras usadas en expresiones
2) Estructuras usadas entre sentencias o grupos de sentencias.
3) Estructuras usadas entre subprogramas.

Estructuras de secuencias de control pueden ser implicitas o explicitas.

Implicitas son aquellas definidas en el lenguaje para que en lo posible sea lo menos modificado por el programador.

Explicitas son aquellas con las que el programador puede opcionalmente modificar la secuencia de operaciones implicitas definidas por el lenguaje, por ejemplo usar parentesis entre expresiones, o

un "goto" referenciando a una etiqueta.

Secuencia de control entre expresiones

Consideremos el siguiente ejemplo

$$ROOT = \frac{-b + \sqrt{b**2-4ac}}{2a}$$

Esta aparente formula simple involucra al menos 15 operaciones separadas en lenguaje de maquina. En ALGOL se codificaria asi:

```
root = (-b+sqrt(b**2-4*a*c))/(2*a)
```

Esta notacion es compacta y el lenguaje procesador concierne con el almacenamiento temporal y la optimizacion. Parece facil decir que la disponibilidad de expresiones en lenguajes de alto nivel es una de las grandes ventajas sobre la maquina y lenguajes ensambladores.

Etiquetas y sentencias "goto"

La habilidad para manejar etiquetas y las transferencias a dichas etiquetas son a menudo indicadas por el uso de la sentencia "goto".

Es conveniente identificar 3 puntos basicos para el uso de sentencias "goto" y de etiquetas :

1) Etiquetas como etiquetas sintacticas, locales durante la traduccion.

2) Etiquetas limitadas en su parte de datos en

tiempo de ejecucion pero sin calculo de etiquetas en el mismo tiempo de ejecucion.

3) Etiquetas sin restricciones en el area de datos durante la ejecucion.

La controversia del GOTO

Desventajas:

- evidencia el diseno de un pobre programa.
- dificultad en el seguimiento del programa
- no permiten mantenimiento rapido del programa.

Ventajas:

- la sentencia "goto" es de proposito generales.
- sin la existencia "goto" una variedad de mecanismos convencionales deben ser cambiados.
- muy usado en la construccion de bloques para simulacion de otras estructuras de control.

9.2.2 SENTENCIAS ITERATIVAS

La estructura basica de una sentencia de una sentencia iterativa consiste de un cuerpo y una cabecera. El cuerpo es compuesto de una secuencia arbitraria de sentencias. La cabeza de una expresion que designa el numero de veces que el cuerpo es ejecutado.

Repeticion simple

Es indicado cuando el cuerpo se quiere ejecutar un numero fijo de veces, ejemplo:

PERFORM body 12 TIMES or

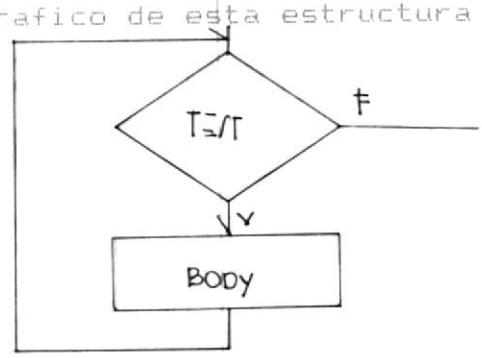
PERFORM body K TIMES

Repeticion condicionada

Su sintaxis es :

WHILE < test > DO < body >

El grafico de esta estructura se muestra:



En esta forma el test es re-evaluado cada vez que termina la ejecucion del cuerpo. Es de suponer que los valores de las variables que aparecen en el test son alteradas.

Repeticion condicionada usando un contador

La cabeza especifica una variable que sirve como contador durante la iteracion.

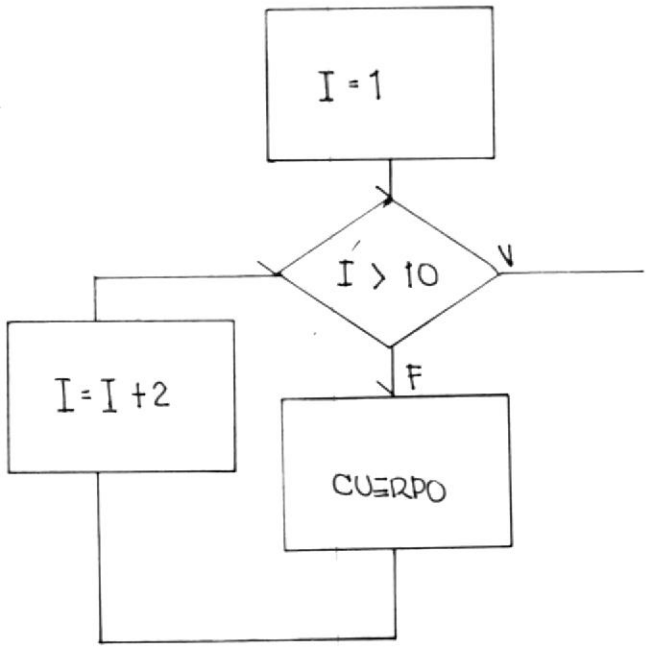
El valor inicial, valor final, y su incremento son especificados en la cabeza y el cuerpo es ejecutado repetidamente usando primero el valor inicial como el valor del indice de la variable mas tantas veces su crecimiento hasta que el valor final es detectado, ejemplo:

```
FOR I = 1 TO 10 STEP 2

        .
        . cuerpo
        .

NEXT I
```

graficamente seria:



9.3 OPERACIONES

Las operaciones se clasifican en dos partes: operaciones definidas por el programador y operaciones definidas por el sistema.

Operaciones definidas por el programador son las usualmente vistas como la suma, resta, multiplicacion, division, raiz cuadrada, exponenciacion, etc. Las operaciones definidas por el sistema seran referenciadas mas adelante segun las conveniencias del lenguaje.

Operaciones elementales

Cada lenguaje tiene un conjunto de elementos constituidos que realiza operaciones basicas

Operaciones aritmeticas

Las operaciones mas primitivas son suma, resta, multiplicacion, division,, raiz cuadrada,

exponenciacion, en la mayoria de los lenguajes. Sus caracteristicas principales son:

1) los operandos y el resultado son numeros y
2) la operacion es una funcion simple.

Operaciones relacionales

Las operaciones relacionales tales como =, <>, >, <, >=, <=, es encontrada en la mayoria de los lenguajes. Estas operaciones toman la forma de simples funciones, aceptando dos numeros como operandos y produciendo un resultado booleano como resultado.

Operaciones Booleanas

Son primitivas en muchos lenguajes. Una operacion booleana solo acepta valores booleanos como operandos y producen un unico valor booleano como resultado. Estas son AND,OR y NOT.

Asignaciones

Asignacion es la operacion basica de modificacion de estructura de datos. Como operandos de asignacion requiere

1) un valor que va a ser almacenado y
2) un apuntador a la direccion en la estructura en la cual el valor va a ser almacenado.

Creacion de estructuras de datos e insercion de elementos

Operaciones que crean nuevas estructuras de datos a traves de la insercion de nuevos elementos son de primaria importancia en el lenguaje.

La creacion de una estructura de datos involucra 4 pasos basicos:

1) creacion de un descriptor para la estructura
2) asignacion de memoria para la estructura misma;
3) especificacion de los valores de los elementos de la estructura;
4) creacion de un camino de acceso a la estructura.

Operaciones de inserciones involucran asignacion de almacenamiento para nuevos elementos, especificando sus valores y algunas veces ajustando los apuntadores luego de la insercion.

Creacion de un camino de acceso a una nueva estructura es realizada a traves de la estructura con un identificador, su nombre, en algunos casos referenciando su situacion o almacenando un apuntador a la estructura ya existente.

9.4 SINTAXIS Y TRADUCCION

Una formal definicion de la sintaxis de un lenguaje de programacion es usualmente llamado gramatica del lenguaje. Una gramatica consiste de un conjunto de definiciones que especifican las secuencias de caracteres que forman programas permisibles en un lenguaje que esta siendo definido. El tipo de gramatica formal mas indicado es el BNF(Backus-naur-form).

Muchos de los elementos mas importantes de la semantica de los programas no son representados en la sintaxis del

programa directamente pero aparecen solamente en forma implicita. Por ejemplo hemos notado el uso de declaraciones implicitas, estructuras de datos implicitas, operaciones implicitas, secuencias de control implicitas.

Propositos generales de la sintaxis

1) Facilidad de lectura

Un programa es "legible" si la estructura fundamental del algoritmo y los datos representados son faciles de entender con una inspeccion rapida del texto del programa. Un programa "legible" es a menudo llamado programa programa "autodocumentado".

2) Facilidad de escritura

Es caracterizado por el uso de una concisa y regular estructura sintactica.

3) Facilidad en su traduccion

4) Carencia de ambiguedad

9.4.1 ELEMENTOS SINTACTICOS DE UN LENGUAJE

Set de caracteres

La seleccion de un conjunto de caracteres es uno de los primeros elementos en ser disenado en la sintaxis para un lenguaje. Determina el numero de caracteres especiales disponibles que pueden ser usados en un programa y datos como delimitadores, simbolos como operadores, y asi por el estilo.

Identificadores

La sintaxis basica para los identificadores(una

cadena de letras y digitos comenzando con una letra).

Simbolos como operadores

La mayoria de los lenguajes usan los caracteres especiales + y - para representar las 2 basicas operaciones aritmeticas. Algunos lenguajes usan una cadena de caracteres para indicar los caracteres especiales arriba mencionados como plus, minus, times, etc.

Palabras claves y reservadas

Una palabra clave es un identificador como parte fija de la sintaxis de una sentencia, por ejemplo: IF, THEN-ELSE, etc. Una palabra clave se convierte en reservada cuando esta no puede ser usada por el programador como un identificador.

Comentarios y palabras significativas

La mayoria de los lenguajes provisionan para comentarios y son encajados en programas como un tipo separado de sentencia y es tratado como sentencia nula.

Palabras significativas son palabras opcionales que pueden ser insertadas en sentencias para improvisar legibilidad. Cobol provee muchas de estas palabras, por ejemplo: la sentencia "goto", el "go" es una palabra clave y el "to" es opcional pero se usa para dar mas comprension al programa.

Espacios

Son de gran variedad en su uso y sirve para separar palabras.

Delimitadores

Es un elemento sintactico usado para marcar el comienzo o el fin de alguna unidad sintactica como una sentencia o una expresion, por ejemplo: begin-end del Pascal.

Libertad y formatos de campo fijo

Una sintaxis es libre si las sentencias de un programa pueden estar en cualquier orden. La mayoria de los lenguajes usan una libertad de sintaxis en forma controlada.

Expresiones

Discutido anteriormente.

Sentencias

Son la mayoria de los componentes en la mayor parte de los lenguajes.

Las principales sentencias son: sentencias estructuradas y las sentencias simples. La sentencia simple es aquella que no contiene otra estructura anidada.

9.4.1 SINTAXIS PARA EXPRESIONES

1) Notacion con prefijo

En esta notacion se escribe el simbolo de la operacion primero, seguido por los operando de izquierda a derecha. Si un operando es el mismo una operacion con operandos, entonces, las mismas

reglas se aplican si tenemos la expresion :

(a + b) * (c - a)

se convierte en

*(+(a,b),-(c,a))

Una variante de esta notacion es la llamada CAMBRIDGE POLISH. En esta notacion los parentesis izquierdos siguiendo a un operador movido para inmediatamente precederlo, y las comas separadoras son eliminadas.

La expresion quedaria :

(*(+ab)(-ca))

Otra variante llamada polish permite sacar los parantesis. Si asumimos que el numero de operandos para cada operador es conocido y fijo, los parentesis son inncesarios. La expresion quedaria :

*(+(a,b),-(c,a))

2) Notacion de postfix

Es similar a la prefija excepto que el simbolo de operacion sigue la lista de operandos. La expresion quedaria :

((a,b)+,(c,a)-)+

o

ab+ca-*

3) Notacion infix

Es adecuada solamente para operaciones binarias por ejemplo las operaciones toman dos operandos.

En la notacion infix el operador es escrito entre dos operandos. Porque notacion para las operaciones basicas aritmeticas, relacionales y logicas, son muy comunmente usadas en las matematicas ordinarias, la notacion para esas operaciones han sido ampliamente adoptadas en un lenguaje.

9.4.3 ETAPAS EN LA TRADUCCION

Principalmente se dividen en 2 partes: el analisis del programa fuente y la sintesis del programa objeto ejecutable.

Analisis del programa fuente

Para el programa analizador el programa fuente aparece inicialmente como una simple cadena de caracteres, luego, esta debe ser analizada caracter por caracter.

El analizador comienza clasificando sus elementos en: identificadores, delimitadores, operadores, numeros, palabras claves, palabras significativas, espacios, comentarios, etc. Esta fase es llamada analisis lexicografico. Tipicamente el analizador lexicografico es la rutina de entrada para el traductor, este debe identificar el tipo de cada item lexicografico, convirtiendo cada numero en una representacion interna de punto fijo o de punto flotante y representando un identificador y almacenandolo en una tabla de simbolos y la

direccion de la tabla de simbolos usada en lugar de la cadena de caracteres.

Analisis sintactico (analizador)

Usualmente alterna con el analisis semantico. Primero el analizador sintactico identifica una secuencia de items lexicograficos formando una unidad sintactica.

Comunmente el sintactico y el semantico se comunican usando un stack. El analizador ingresa en el stack los varios elementos de la unidad sintactia y luego son recobrados por el analizador semantico.

Analizador semantico

Las estructuras sintacticas reconocidas por el analizador sintactico son procesadas y la estructura de la fase del codigo objeto ejecutable comienza a tomar forma.

El analizador semantico puede producir el codigo objeto ejecutable en simple traduccion. Pero mas comunmente la salida de esta etapa es alguna forma interna del programa final ejecutable que es entonces manejado por la fase de optimizacion del traductor antes de generar el programa objeto.

Las mas comunes de las funciones pueden ser descritas, pero las principales son:

1) Mantenimiento de la tabla de simbolos

Una tabla de simbolos es una de las estructuras de

datos central en cada traductor contiene una entrada para cada identificador diferente encontrado en el programa fuente. El analizador lexicografico toma la entrada inicial conforme lo encuentra en el programa fuente, pero el analizador semantico toma responsabilidad despues. La tabla de simbolos contiene atributos acerca de sus identificadores: su tipo(variable simple, arreglo, etc), su clase de valor(entero o real), etc. El analizador semantico entra esta informacion en la tabla conforme se procesan las declaraciones, subprogramas, etc.

2) Insercion de informacion implicita

A menudo en el programa fuente, la informacion es implicita que debe ser realizada explicita en un mas bajo nivel en el programa objeto. La mayoria de estas informaciones implicitas van bajo el nombre de convenciones asumidas, estas convencionalidades pueden ser descartadas por declaraciones explicitas del programador.

3) Detencion de errores

El analizador semantico no solamente debe reconocer como error a aquellos que producen un apropiado mensaje de error, tambien debe determinar la forma apropiada para continuar con el analisis del resto del programa.

9.5 EL LENGUAJE BASIC 1.0

Este lenguaje fue disenado con propositos educativos, es decir, facilmente programable, para personas con algun o sin ningun conocimiento de programacion. Es un lenguaje escrito para que sea programado con instrucciones en espanol.

Este lenguaje se compone de 2 partes : el programa compilador y el programa interpretador.

El compilador trata de analizar sintacticamente cada sentencia y determinar los errores de sintaxis a lo largo del programa fuente.

El programa interprete se encarga de generar el codigo objeto y de ejecutarlo.

Ambos programas son escritos en lenguaje Pascal y desarrollado en un computador "APPLE".

9.5.1 COMO OPERA EL BASIC 1.0

Este lenguaje solo maneja variables enteras y reales. No maneja variables alfanumericas para ningun caso.

Tiene formato libre para escribirlo, es decir, que una instruccion puede usar una o mas lineas.

Establecemos una comparacion entre este Basic y el Basic normal que conocemos en cuanto a su conjunto de instrucciones :

aBASIC 1.0	BASIC NORMAL
Asigne	Let

Si-Luego-Sino	If-Then-Else
Com(comentario)	Rem
Digite	Input
Muestre	Print
Fin	End
Parrafo "xxxx"	"Usa etiqueta num."
Vaya "xxxx"	Goto "9999"(num)
Mientras	No existe (equivale WHILE)
Finsi	Endif
Finlazo	No existe (equivale ENDWHILE)

Estructuras de Datos Usadas

Como solamente emplea variables numericas no existe estructura compleja alguna.

El programa compilador usa un stack para almacenar los nombres de los identificadores de manera que se pueda identificar un nombre de campo de una palabra reservada y de un nombre de procedimiento.

Los nombres de variables enteras como para las reales reales son indiferentes.

Internamente el programa lo maneja todo como numeros reales, es decir, que por mas que empleen variables enteras, internamente se efectuaran operaciones con variables de tipo real.

Cada vez que se detecta un campo, se chequea si este nombre existe en la tabla de procedimientos, si no esta, entonces el compilador lo ingresa en su stack, siempre que no lo haya ingresado

anteriormente.

Estructuras de control Usadas

Las estructuras de control usadas son : SI-LUEGO-SINO, MIENTRAS-FINLAZO, VAYA.

SI-LUEGO-SINO

Actua de manera semejante al If-Then-Else del Basic ordinario. Su sintaxis se define asi:

```
SI   <condicion>
 LUEGO
            .
            .  cuerpo
            .
 SINO
            .
            .  cuerpo
            .
 FINSI
```

El delimitador final del "si" es "finsi", ya sea que se use "sino" o no, y es de caracter obligatorio.

El "cuerpo" puede contener una o mas sentencias.

Se permiten "si" anidados sin ningun problema, y tambien estructuras de control incluidas.

Su formato de escritura es libre.

MIENTRAS-HAGA-FINLAZO

Actua de manera similar a la sentencia "while".

Su formato es :

MIENTRAS <condicion>

HAGA

.
. cuerpo
.

FINLAZO

El "cuerpo" puede tener una o varias sentencias.

El delimitador es "finlazo" y es obligatorio.

Permite la inclusion de otras estructuras de control en forma anidada.

VAYA

Funciona de manera similar al "goto" del Basic ordinario. Su formato es :

VAYA <etiqueta>

donde etiqueta es nombre alfanumerico, a diferencia del basic ordinario donde su etiqueta es un numero de sentencia.

Esta sentencia esta ligada a la sentencia no ejecutable "parrafo". Su formato es :

PARRAFO <nombre procedimiento>

donde el nombre del procedimiento esta conectado con una sentencia "vaya" sea que este o no definida todavia.

Para llevar un control seguro de que cada "vaya" este ligada con su "parrafo" correspondiente, se usan dos stacks : el uno para almacenar las etiquetas nombradas en cada "vaya" y el otro

almacena las etiquetas definidas en "parrafo".

Al fin de la compilacion se confrontan ambas tablas y se emitiran errores si algun "vaya" no encuentra su pareja.

Tambien chequea que cada etiqueta no haya sido usada como campo.

Sentencias de E/S

Las dos sentencias de entrada/salida definidas para este lenguaje son : Digite (Input) y Muestre (Print); solo concierne a la pantalla tanto la entrada como la salida de datos. Su formato es :

DIGITE <campo1, campo2, ...campo n>

MUESTRE <campo1, campo2, ...campo n>

BIBLIOTECA

9.5.2 EL PROGRAMA INTERPRETE

Este programa se encarga de generar el codigo objeto y de ejecutarlo.

Generacion de Codigo Objeto

Cada numero o campo genera su direccion como atributo.

A continuacion se analizaran las sentencias principales en lenguaje ensamblador usados en la generacion del programa objeto :

```
LIT  0,a     carga una constante
OPR  0,a     ejecuta una operacion
LOD  l,a     carga la variable l,a
STO  l,a     almacena la varialbe l,a
JMP  0,a     bifurca a "a"
```

```
JPC  0,a     bifurcacion condicional  a
                "a"
LEE   0,a     lee la linea de datos  en
                "a" numero de variables
FLN    0,0    salta  a  la  siguiente
                linea de escritura
FIN    0,0    Fin de ejecucion
```

Por ejemplo si tuvieramos el siguiente fragmento de un programa en PL/I :

```
CONST   m=7,   n=85
VAR     x, y, z, a, r
PROCEDURE multiply;
  VAR  a, b;
  BEGIN
     a:=x;
     b:=y;
     z:=0;
     WHILE b > 0 DO
       BEGIN
          IF odd b THEN z:=z+a;
          a:=2*a;
          b:=b/2
       END
  END;
```

La generacion de su correspondiente codigo objeto

```
seria
                3  LOD  1,3   x
                4  STO  0,3   a
                5  LOD  1,4   y
                6  STO  0,4   b
                7  LIT  0,0   0
                8  STO  1,5   z
                9  LOD  0,4   b
               10  LIT  0,0   0
               11  OPR  0,12  >
               12  JPC  0,29
               13  LOD  0,4   b
               14  OPR  0,7   odd
               15  JPC  0,20
               16  LOD  1,5   z
               17  LOD  0,3   a
               18  OPR  0,2   +
               19  STO  1,5   z
               20  LIT  0,2   2
               21  LOD  0,3   a
               22  OPR  0,4   *
               23  STO  0,3   a
               24  LOD  0,4   b
               25  LIT  0,2   2
               26  OPR  0,5   1
               27  STO  0,4   b
               28  JMP  0,9
               29  OPR  0,0   return
```

Nota.- La operacion "odd" no es una funcion usada en el Basic 1.0. Solo se la usa para un mero ejemplo.

Como se puede ver el sistema usado para evaluar expresiones aritmeticas es el paso de Infix-Postfix.

Con un ejemplo sencillo quedara finalmente ilustrado :

ASIGNE C = A - B

LOD	0,4	carga A (en primera entrada)
LOD	0,5	carga B (en segunda entrada)
OPR	0,3	reste A-B (resta primera de segunda entrada)
STO	0,6	asignelo a C (primera entrada se almacena en la direccion asignada a C)

Estructuras de Almacenamiento

Un stack almacena las instrucciones en codigo objeto y luego sera ejecutada conforme se lo este accesando. El formato de esta instruccion objeto es :

| F | l | a |

donde F es el codigo de operacion, y un parametro que consiste de una o dos partes : en el caso de operadores el parametro "a" determina la identidad del operador; en otros casos un numero, la direccion de una bifurcacion, o la direccion de un dato.

Un stack para almacenamiento de datos. Para llegar a la direccion correcta dentro del stack tenemos primero que accesar al stack de instrucciones en codigo objeto de la que obtendremos la direccion en el stack de datos. Este stack tiene separada sus 3 primeras entradas para almacenamiento de resultados temporales de operaciones aritmeticas quedando el resto del stack para almacenar datos del usuario.

Estructuras de Control

El problema critico de las estructuras de control es la actualizacion de la direccion de bifurcacion. A continuacion las analizaremos.

Si-Luego-Sino-Finsi

La condicion sera evaluada antes de cualquier cosa. De esta manera si el resultado fuera verdadero ejecutara las instrucciones que sigan la palabra "luego" y saltara las que estuvieran encabezadas por "sino" hasta encontrar la primera instruccion luego del "finsi".

Si fuera falso el resultado de la evaluacion de la condicion debera ejecutar las sentencias que estuvieren encabezadas por "sino".

Mientras-Haga-Finlazo

Se evaluara la condicion antes de entrar al lazo. Si no se cumple se hara un salto a la siguiente instruccion luego del "finlazo".

Vaya

Al encontrarse con esta sentencia se buscara el nombre del parrafo dentro de la tabla donde se tiene los identificadores.

Variable	localizacion de memoria asignada
Etiqueta	direccion a la que debera ir para continuar la ejecucion del programa

De aqui se obtendra la direccion que se le colocara a la instruccion "jump" que equivale a la instruccion "vaya". En caso que no exista ese parrafo definido en la tabla, debera ser

almacenado temporalmente en una tabla con la posicion dentro de la tabla de codigos que debera ser actualizada con la direccion de la etiqueta del parrafo.

En el momento que se encuentre con esta sentencia debera ingresar ese nombre de parrafo, si es que no existiera, y la posicion del indice del codigo objeto que en ese momento tiene. Ademas debera leer por cada nuevo nombre, todas las entradas de aquella tabla temporal a las que debera actualizar la direccion de la etiqueta que no habia encontrado.

10. CONCLUSIONES Y RECOMENDACIONES

Este lenguaje, orientado a principiantes, siendo muy limitado tanto en sus estructuras de datos como sus estructuras de control, por esto consideramos que aun se podria mejorar para dar mas facilidades en su uso.

Hacemos algunas observaciones en cuanto a mejoras y a su perfeccionamiento:

1) manejo de caracteres alfanumericos
2) soporte de tablas y arreglos tanto numericos como alfanumericos
3) estructruas de control con varias opciones de condicionamiento, por ejemplo como instruccion PERFORM del COBOL
4) manejo de archivo de datos
5) manejo de arreglos y tablas

Hemos mencionado las principales mejoras que podrian ser implementadas.

Luego de esto la intencion de construccion de un lenguaje es que tambien tenga su propio sistema operativo y un editor de texto.

La construccion de un sistema operativo es un topico distinto a un lenguaje pero de igual forma interesante. Se puede partir de que el lenguaje BASIC 1.0 se ha escrito en pascal, por lo tanto el sistema operativo debera ser compatible, y lo mismo sucede con la

elaboracion del editor de texto.

11. BIBLIOGRAFIA

- Algorithms + Data Structures = Programs

  Autor : Nicklaus Wirth

- Programming Languages : Design and Implementation

  Autor : Terrence W. Pratt

- Compiler Design Theory

  Autor : Lewis - Rosenkrantz - Stearns


BIBLIOTE