Proyecto Lenguaje de Programación compiladores e intérpretes UNET 2011

Se solicita la creación de un interprete (analizador léxico, sintáctico, semántico y ejecución) usando la herramientas jFlex que a su vez haga uso del analizador sintáctico creado con CUP (ambos vistos en clase) para llevar a cabo el proceso de análisis léxico, sintáctico y el AST (incluyendo una tabla de símbolos) del lenguaje descrito en la parte inferior (alguna duda de la especificación por favor SOLO preguntar en el grupo: http://groups.google.com/group/compiladores_e_interpretes).

Notas:

- ***A manera de recordatorio, el lenguaje de implantación debe ser Java.***
- ***La fecha máxima de entrega es el 03/11/2011 (IMPRORROGABLE)***
- ***Grupos de hasta 4 personas la, evaluación individual (tendrá mucho MUCHO más peso que grupal), todos deben conocer como se hizo todo, se validara a cada persona su conocimiento de la aplicación y debido a problemas en semestres anteriores cualquier problema de copia o violación a una norma de ética simple acarreara la colocación de cero puntos en el compilador y por lo tanto la perdida de la nota del último parcial de la materia que es la construcción del compilador.***
- ***Se aconseja usar un repositorio GIT (como github y una herramienta como tortoisegit en windows) para trabajar de forma distribuida y llevar a cabo un buen control de versiones sin traumas y sin problemas (según entiendo estos repos pueden ser accedidos dentro de la UNET incluso sin configurar proxy). Recuerde no está en la edad de piedra así que no haga versiones en una carpeta en una usb y luego sufra las consecuencias de su acción.***
- ***Por último se les aconseja seguir el consejo anterior, aunque esto les puede hacer perder casi 3 HORAS de su vida leyendo dos páginas/tutoriales en internet que explique su uso, para que:***
    - ***Aprendan algo nuevo y útil en su carrera y que además es muy usado a nivel mundial si quieren ser programadores "serios" algún dia.***
    - ***Hagan su vida más fácil mediante el uso de las herramientas correctas y dejen de vivir en la era del usb.***

Las Fuentes para la especificación del intérprete mostrada a continuación fueron:
http://cfievalladolid2.net/tecno/cyr_01/control/lengua_BASIC/index.htm
http://www.aprendeaprogramar.com//course/view.php?id=6

INTRODUCCION

BASIC es un lenguaje de programación de propósito general que ofrece economía sintáctica, control de flujo, estructuras sencillas y un buen conjunto de operadores. Es un lenguaje que no está especializado en ningún tipo de aplicación. Esto lo hace un lenguaje versátil y potente, con un campo de aplicación ilimitado y, sobre todo, se puede aprender rápidamente. En poco tiempo, un programador puede utilizar la totalidad del lenguaje.

La palabra BASIC proviene de la expresión inglesa ***B**eginner's **A**ll-purpose **S**ymbolic **I**nstruction **C**ode*: código de instrucciones simbólicas de propósito general para principiantes.

El BASIC fue el primer lenguaje de programación desarrollado. Lo fue a mediados de la década de los sesenta por los profesores John G. Kemeny y Thomas E. Kurtz en el Dartmouth College, en California. Su código se basa en el vocabulario inglés y en las expresiones matemáticas. Consta de cientos de instrucciones para utilización de gráficos, procesado de textos, uso de variables, archivos, etc. EL BASIC utiliza un alfabeto formado por los caracteres alfabéticos: A-Z, cifras 0-9, caracteres especiales como operadores aritméticos: +, -, *, etc., y otros: (,),$, etc.

FUNDAMENTOS

La mejor forma de aprender un lenguaje es programando con él. El programa más sencillo que se puede escribir en BASIC es el siguiente:

```

```

Como podemos imaginar, este programa sin una sola instrucción no hace nada, pero se interpreta correctamente y nos da una idea de que no se necesita mucho para empezar a programar en BASIC. Esto no es cierto en otros lenguajes como C, en los que hay que utilizar obligatoriamente encabezados de programa y limitadores de bloques de sentencias.

Un programa algo más complicado, pero que hace algo, es el siguiente:

```
' Este es mi primer programa en BASIC

PRINT "Bienvenido a la programación en lenguaje BASIC"

END
```

Con él visualizamos en la pantalla el mensaje:

```
Bienvenido a la programación en lenguaje BASIC
```

La instrucción `PRINT`. Ésta toma como argumento una cadena de caracteres limitados por dobles comillas `" "` y la imprime en la salida habitual, que generalmente es la pantalla del PC en el que trabajamos.

La instrucción `END` termina el programa. Ni siquiera es obligatorio finalizar con `END`, aunque es conveniente por una cuestión de claridad. De cualquier manera, si extraemos esa sentencia, el programa funcionará exactamente igual (el programa finaliza automáticamente cuando no se encuentran más líneas de código).

La inclusión de comentarios en un programa es una saludable práctica, como lo reconocerá cualquiera que haya tratado de leer un listado hecho por otro programador o por sí mismo, varios meses atrás. Para el intérprete los comentarios son inexistentes, por lo que no generan acciones a ejecutar, permitiendo abundar en ellos tanto como se desee. En el lenguaje BASIC se toma como comentario todo carácter que sigue a la comilla simple: `'` o a la palabra clave `REM`.

Como se observa, un programa en Basic es simplemente un archivo de caracteres que contiene un conjunto de instrucciones que un programa especial, el **intérprete**, se encarga de transformar en un código que la computadora puede ejecutar (aunque no se genera un archivo .EXE).

Antes de continuar hagamos un inciso sobre la nomenclatura utilizada en Basic para designar a distintos elementos del lenguaje:

Elemento	Definición
Sentencia Instrucción (Comando)	Es una instrucción que realiza una operación. Se utilizan estas palabras de forma intercambiable. Se usa mucho la forma *comando* debido a su utilización en inglés.
Palabra clave *Keyword*	Una palabra que forma parte del lenguaje Basic. Cuando se utiliza una palabra clave en una sentencia, Basic automáticamente la convierte a mayúsculas. No se puede utilizar una palabra clave para dar nombre a una constante o variable (palabra reservada).
Cadena *String*	Una cadena de texto (caracteres). En Basic se delimita con comillas, por ejemplo, "Bienvenido". La cadena "" (vacía) es válida.
Número	Designa a números enteros, decimales de precisión simple y decimales de precisión doble.
Constante	Un valor (cadena o número) que no cambia a lo largo de la ejecución del programa. Hay dos clases de constantes; las **constantes literales** están escritas directamente en el código: "Hola", 16; las **constantes simbólicas** o con nombre son valores constantes a los que se les asigna un nombre. Por ejemplo, LONG_MAX en lugar de 16. Las constantes simbólicas se definen usando la palabra clave CONST.
Variable	Un contenedor con nombre para un número o una cadena. Son el medio para que el programa "recuerde" datos. Se pueden crear

	variables de varias maneras: calculando un valor, tomando una entrada de usuario, leyendo un archivo, etc.
Operador	Designa una operación matemática. Los operadores pueden ser aritméticos, relacionales o lógicos.
Función	Devuelve una cadena o un número. Puede tomar uno o más parámetros de entrada para calcular el resultado.
Comentario	No hace nada en el programa. Se utiliza como una nota explicativa que contribuye a clarificar el código y a recordar más tarde lo que se hizo.
Bloque	Designa a un conjunto de líneas dentro de la estructura del programa, que guardan una relación de algún tipo entre sí.
Bucle	Designa a un grupo de líneas que se ejecutan una serie de veces.

La estructura general de un módulo de código en Basic consta de un programa principal y de varios subprogramas y/o funciones (***PARA EFECTOS DEL PROYECTO EXISTIRAN PROCEDIMIENTOS PERO NO SE PODRAN EJECUTAR DE FORMA RECURSIVA*)*:**

```
' Programa principal

        declaración de funciones y subprogramas
        declaración de variables
        sentencias
END
SUB Subprograma1 ( )

        declaración de variables
        sentencias

END SUB
FUNCTION Funcion1 ( )

        declaración de variables
        sentencias

END FUNCTION
...
...
...

SUB Subprograman( )

        declaración de variables
        sentencias
END SUB
FUNCTION Funcionn ( )

        declaración de variables
        sentencias

END FUNCTION
```

VARIABLES Y TIPOS BASICOS

En un programa BASIC es conveniente ***(OBLIGATORIO EN NUESTRO PROYECTO)*** definir todas las variables que se utilizarán antes de comenzar a usarlas, a fin de indicarle al intérprete de que tipo serán y, por tanto, cuánta memoria debe destinar para albergar a cada una de ellas. Veamos un ejemplo:

```
Multiplica dos números enteros

          DIM multiplicador AS INTEGER ' defino <multiplicador> como un entero
          DIM resultado AS INTEGER     ' defino <resultado> como un entero

          multiplicador = 1000         ' asigno valores
          resultado = 2 * multiplicador

          PRINT "Resultado = ", resultado       ' muestro el resultado
END
```

Así, una variable es un lugar donde se puede almacenar temporalmente un dato. En BASIC las variables tienen un nombre que las identifica, y sirve para hacer referencia a ellas. También tienen un tipo, que es el tipo de datos que puede almacenar. El valor de las variables es, como su propio nombre indica, variable. Podemos alterar su valor en cualquier punto del programa.

Por efectos de simplificaciones ***nuestro proyecto solo tendrá variables del tipo INTEGER***.

Para dar un nombre a una variable tenemos que usar un **identificador**. La longitud de un identificador puede variar entre uno y varios caracteres, por lo general, 40. En la mayoría de los casos el primer carácter debe ser una letra. A continuación se muestran varios ejemplos de nombres de identificadores correctos e incorrectos:

Correcto	Incorrecto
`cuenta`	`1cuenta`
`prueba23`	`prueba*`
`puerto.paralelo`	`puerto_paralelo`

El lenguaje Basic **no es sensible a mayúsculas y minúsculas** (no es *case sensitive*), de modo que para el intérprete es lo mismo el identificador `cuenta` que otro denominado `Cuenta`.

Los intérpretes y compiladores reservan determinados términos ó palabras claves (*keywords*), para el uso sintáctico del lenguaje, tales como: `CLS`, `PRINT`, `END`, etc. y no se pueden utilizarla en nombres de variables.

Para crear una variable en un lugar determinado del un programa escribiremos primero el tipo de variable y luego el identificador con el que queremos nombrar la variable. A esto se le denomina **definir una variable**. La forma general de la definición es:

```
DIM identificador [AS tipo]
DIM identificador[sufijo]
```

Por ejemplo:

```
DIM numero AS INTEGER       ' crea la variable numero, de tipo número entero
```

En Basic, las variables no se pueden inicializar (es decir, establecer un valor inicial) en el momento de creación. Por ejemplo, es incorrecto:

```
DIM numero% = 0         ' incorrecto, falla el intérprete
DIM frase$ = "Hola"     ' incorrecto, falla el intérprete
```

Aunque la asignación de valores a las variables hay que realizarla posteriormente a la definición de las mismas.

Veamos el siguiente programa de ejemplo acerca del uso de variables:

```
' Convierte grados Centígrados a Fahrenheit
       DIM cels AS INTEGER, fahr AS INTEGER

       cels = 25                      ' Temperatura en ° C
       fahr = 32 + 9 * fahr / 5       ' Temperatura en ° Fahrenheit

       PRINT ">>> ";cels;" °C son ";fahr;" °F

END
```

En él se definen dos variables `single`, se asigna un valor a la primera y se calcula la segunda mediante una expresión aritmética. En la instrucción `PRINT`, el ; indica que se concatenen los valores a mostrar.

Es de hacer notar que es un salto de línea lo que separa cada instrucción de otra del lenguaje a diferencia de lenguajes como C, donde el símbolo de separación de sentencias es el “;” por lo que el salto de línea no podrá ser ignorado como ha sido lo habitual sino será un token o testigo del analizador léxico para denotar fin de sentencia.

Es de hacer notar que nuestro proyecto manejara vectores, que serán manejados al estilo lenguaje C, y se declararan igual que las variables pero tendrán la forma: vector[tamaño_maximo], e igualmente los tipos de datos a usar serán Entero, Real y Cadena (cadena no será usado en vectores)

Funciones para E/S de datos

Unas de las principales vías de comunicación de un programa con el usuario son la pantalla (terminal) y el teclado. La entrada y salida de datos por estos medios se pueden realizar mediante varias instrucciones de Basic. Las fundamentales son:

`PRINT: Salida por pantalla`

Comenzaremos con la instrucción principales de salida de datos: `PRINT`.

La función `PRINT` escribe texto y/o números en la pantalla (aunque también en un archivo). Su uso en sencillo, sólo hay que añadir detrás de la palabra clave la lista de datos que se desea visualizar. Por ejemplo:

```
' Muestra un mensaje

        PRINT "Hola"
        PRINT "mundo"
END
```

El resultado es:

```
Hola
mundo
```

Nótese que PRINT imprime un salto de línea tras el texto. Se puede inhibir el salto automático de línea si se añade ; tras el texto:

```
' Muestra un mensaje

        PRINT "Hola";
        PRINT "mundo"
END
```

Cuyo resultado es:

```
Hola mundo
```

Un ejemplo de uso con valores numéricos y de texto es:

```
' Muestra valores numéricos

        a = 50
        b = 100

        PRINT "El valor de a es"; a; " y el valor de b es"; b

END
```

cuyo resultado es:

```
El valor de a es 50 y el valor de b es 100
```

Nótese que al concatenar el texto con los números mediante ; se añade automáticamente un espacio en blanco antes de cada valor numérico.

Si en lugar de utilizar ; (punto y coma) se utiliza , (coma) para la concatenación:

```
 ' Muestra valores numéricos

        a = 50
        b = 100

        PRINT "El valor de a es", a; " y el valor de b es", b

END
```

el resultado es:

```
 El valor de a es              50 y el valor de b es             100
```

es decir, produce una separación de 14 espacios. ***(Esto no lo debería permitir su compilador debería marcarlo como un error en lugar de crear la separación de 14 espacios)***

INPUT: lectura del teclado

Algo muy usual en un programa es esperar que el usuario introduzca datos por el teclado. Para ello contamos con la instrucción INPUT. Un ejemplo:

```
' Lee un número entero desde el teclado
        DIM num%

        PRINT "Introduce un número";
        INPUT num%

        PRINT "Has tecleado el número"; num%
END
```

Este programa muestra el mensaje `Introduce un número?` (nótese la interrogación añadida) y espera a que el usuario introduzca un entero. Si el usuario escribe `23` y pulsa `INTRO` lo que se observa es:

```
Introduce un número? 23
Has tecleado el número 23
```

Es muy común escribir un mensaje con PRINT antes de realizar un INPUT, como en los dos ejemplos anteriores. Pero INPUT facilita la labor si se utiliza así:

```
' Lee un entero desde el teclado
        DIM num%

        INPUT "Introduce un número: ", num%

        PRINT "Has tecleado el número"; num%
END
```

que produce la salida:

```
Introduce un número: 23
Has tecleado el número 23
```

NOTAS:

- ***En su intérprete siempre se usara la declaración completa de variables de BASIC (incluyendo la palabra INTEGER , FLOAT o STRING aunque en los ejemplos aquí mostrados a veces no lo hagan).***
- ***Solo se podrán declarar variables de tipo entero, real y cadena e igualmente solo se imprimirán y leerán este tipo de variables (incluyendo vectores de enteros o reales).***
- ***Es de recalcar que debido a que este lenguaje no finaliza sus sentencias en ; el token o testigo de finalización de línea o sentencia en su compilador debería ser el salto de línea.***
- ***Además aunque las versiones de Basic aquí vistas colocan un numero al inicio de la línea nosotros en nuestra versión no lo haremos.***
- ***Igualmente cada línea se limitara a una sentencia por línea evitando usar el operador de ":" para múltiples sentencias en una línea.***

Comentarios

A veces nos puedes interesar que en nuestro programa aparezcan comentarios, que a nosotros nos aclaren por qué hemos dado un cierto paso. Será muy útil cuando se trabaje en grupo, o incluso cuando nosotros mismos tengamos que revisar un programa un cierto tiempo después de haberlo creado: nos ahorrará tiempo de "descifrar" qué habíamos hecho y por qué.

Los **comentarios en Basic** se indican con la palabra REM (abreviatura de REMARK, comentario, en inglés):

```
REM SIMPLEMENTE UN SALUDO
PRINT "HOLA"
```

La gran mayoría de las versiones del lenguaje Basic permitirán abreviar todavía más la palabra REM, poniendo un apóstrofe (') en su lugar:

```
 ' SIMPLEMENTE UN SALUDO
 PRINT "HOLA"
```

Vamos a ver un ejemplo, que pida dos números al usuario y muestre su suma, pero ya escrito con un poco más de corrección:

```
' Sumar dos numeros que introduzca el usuario
PRINT "Introduce los dos numeros que quieres sumar"
INPUT "Primer numero"; x
INPUT "Segundo numero"; y
PRINT "Su suma es "; x+y
```

Los bucles for

La palabra "**bucle**" es el nombre que se da a una parte del programa que se repite un cierto número de veces. En Basic, esto se consigue con la orden **FOR**, que vamos a ver directamente con un ejemplo:

```
' Escribimos un saludo 10 veces
 FOR i = 1 TO 10
 PRINT "Hola"
 PRINT "Como estas?"
 NEXT i
 PRINT "Se acabaron las repeticiones"
```

En español, este FOR se podría traducir como un "Desde", de modo que la línea 20 sería "desde que 1 valga 1 hasta 10", es decir estamos usando una variable auxiliar, que comenzará valiendo 1, y el trozo de programa se repite lo hará hasta que esa variable i valga 10.

La línea 50 indica el final de la parte que se repite, y hace que se pase al próximo (next) valor de la variable i. Es decir, hace que pase de 1 a 2 y que se vuelvan a repetir las líneas 30 y 40; en la siguiente pasada, i aumenta de 2 a 3, y se vuelven a repetir las líneas 30 y 40; así sucesivamente, hasta que i vale 10, momento en que dejan de repetirse las líneas 30 y 40.

El resultado de este programa será:

```
Hola
Como estas?
Hola
Como estas?
Hola
Como estas?
Hola
Como estas?
Hola
Como estas?
Hola
Como estas?
Hola
Como estas?
Hola
Como estas?
Hola
Como estas?
Hola
Como estas?
Hola
Como estas?
Se acabaron las repeticiones
```

Es muy frecuente emplear la orden FOR para contar, aprovechando eso de que la variable empieza valiendo 1 y termina valiendo lo que le indiquemos. Por ejemplo, podemos contar del 1 al 8 así:

```
 ' Escribimos del 1 al 8
 FOR n = 1 TO 8
 PRINT n
 NEXT n
```

que escribiría lo siguiente en pantalla:

```
1
2
3
4
5
```

```
6
7
8
```

O podemos escribir la tabla de multiplicar del 5 así:

```
' Tabla de multiplicar del 5
FOR factor = 1 TO 10
PRINT "5 x "; factor; " vale: ";
PRINT factor * 5
NEXT factor
```

El resultado de este programa sería:

```
5 x 1 vale: 5
5 x 2 vale: 10
5 x 3 vale: 15
5 x 4 vale: 20
5 x 5 vale: 25
5 x 6 vale: 30
5 x 7 vale: 35
5 x 8 vale: 40
5 x 9 vale: 45
5 x 10 vale: 50
```

No necesariamente tenemos que contar **de uno en uno**. Para ello se emplea la orden **STEP** (paso, en inglés). Por ejemplo, podemos escribir los número pares del 1 al 20 así:

```
' Numeros pares del 1 al 20
FOR n = 2 TO 20 STEP 2
PRINT n
NEXT n
```

O podemos contar "hacia atrás" si usamos un paso negativo:

```
' Descontar desde 10 hasta 0
FOR x = 10 TO 0 STEP -1
PRINT x
NEXT x
```

Podemos **incluir un FOR dentro de otro**. Por ejemplo, podríamos escribir la tabla de multiplicar completa así:

```
' Tabla de multiplicar completa
FOR tabla = 1 to 10
FOR factor = 1 TO 10
PRINT tabla; " x "; factor; " vale: ";
PRINT tabla * factor
NEXT factor
PRINT : ' Linea en blanco entre tablas
NEXT tabla
```

En este programa he ido escribiendo un poco más a la derecha lo que dependía de cada FOR, para intentar que resulte más legible. Esto es lo que se conocer como "**escritura indentada**", y yo lo comenzaré a emplear a partir de ahora.

El resultado de este programa debería ser algo así:

```
1 x 1 vale: 1
1 x 2 vale: 2
1 x 3 vale: 3

[... (muchas mas lineas aqui en medio) ...]

9 x 7 vale: 63
9 x 8 vale: 72
9 x 9 vale: 81
9 x 10 vale: 90

10 x 1 vale: 10
10 x 2 vale: 20
10 x 3 vale: 30
10 x 4 vale: 40
10 x 5 vale: 50
10 x 6 vale: 60
10 x 7 vale: 70
10 x 8 vale: 80
10 x 9 vale: 90
10 x 10 vale: 100
```

La condición IF y operadores lógicos AND, OR, NOT

Los Basic modernos, que trabajan sin números de línea, suelen permitir otra forma ampliada:de escribir la orden IF, que permite dar varios pasos si se cumple la condición o si no se cumple, y que termina con **ENDIF**:

```
IF condición THEN
lista_de_ordenes_1
ELSE
lista_de_ordenes_2
ENDIF
```

Hemos empleado el signo ">" para ver si una variable es mayor que un cierto valor. Veamos cómo se indican las otras **comparaciones** posibles:

```
> Mayor que
>= Mayor o igual que
< Menor que
<= Menor o igual que
= Igual a
<> Distinto de
```

Las condiciones se pueden enlazar con **AND** (y), **OR** (ó), **NOT** (no), pero entonces puede ser interesante emplear paréntesis, para que el resultado sea más legible (y para evitar problemas de precedencia de operadores, que no trataremos todavía), por ejemplo:

```
IF ((a>2) AND (b<>3)) OR (NOT (c>3)) THEN PRINT "Es
valido"

PRINT "Introduzca un numero"
INPUT x
IF x > 10 THEN

PRINT "Mayor de 10"

ELSE

PRINT "Menor o igual que 10"

ENDIF
```

Otros bucles - While, Loop...

Hemos visto como emplear la orden FOR para hacer que una parte de un programa se repita incondicionalmente. También hemos comentado cómo comprobar condiciones con la orden IF.

Pero en la práctica es muy frecuente que nos encontremos una "mezcla" de ambas cosas: una serie de operaciones que se deben **repetir mientas se cumpla una cierta condición**.

La primera es la orden **WHILE**, cuyo formato es

```
WHILE condición
órdenes
WEND
```

De modo que podríamos reescribir nuestros programas de una forma mucho más legible:

```
CLS
WHILE clave <> "xjk5"
PRINT "ESCRIBE TU CLAVE DE ACCESO"
INPUT clave
WEND
```

La segunda instrucción es el ciclo repita hasta que:

(se repite HASTA QUE se cumpla la condición, pero esa condición se comprueba lo primero de todo)o también:

```
DO
órdenes
LOOP UNTIL condición
```

```
CLS
```

DO

```
PRINT "ESCRIBE TU CLAVE DE ACCESO"
INPUT clave
LOOP UNTIL clave == "xjk5"
```

Operadores

Aritméticos.

Todos aquellos símbolos utilizados en una expresión aritmética cuyos operandos son variables o constantes de tipo numérico.

Operador	Significado
+	Adición
-	Sustracción
*	Multiplicación
/	División
^	Potenciación
Mod	Resto de la División Entera

Relacionales.

Son todos aquellos símbolos utilizados para establecer una relación entre variables del mismo tipo, cuyo resultado final es un valor del tipo lógico (Verdadero o Falso).

Operador	Significado
=	Similitud
>	Mayor que
<	Menor que
<=	Menor o igual
>=	Mayor o igual
<>	Diferente

Lógicos.

Se utilizan en conjunto con los operadores relacionas para formar las llamadas expresiones lógicas, permitiendo la formación de nuevas proposiciones lógicas a partir de otras más simples.

Operador	Significado
Y	Conjunción Lógica
O	Disyunción Lógica
NO	Negación Lógica

Nota:

Las expresiones tendrán la asociatividad, precedencia y prioridad de operadores acostumbrada

Prioridad de los operadores

Al escribir expresiones se encontraran ocasiones en las cuales se utilice más de un operador, y dependiendo del orden en que se efectúen las operaciones el resultado puede verse afectado. Para evitar este problema se establece un orden de precedencia entre los operadores que nos indica el orden en que serán efectuadas las operaciones.

Prioridad	Operadores
1	Paréntesis
2	Funciones (no tenemos)
3	^
4	*, /
5	+, -
6	=, <, <=, >, >=, <>
7	NO
8	Y
9	O

Notas finales

- Crearemos un intérprete.
- El lenguaje puede ser adaptado y esta especificación es solo una guía de las directrices principales del mismo, pero cualquier cambio que le añada claridad o mejore algún aspecto del mismo será permitido.
- Como mínimo su lenguaje manejara procedimientos (sin recursividad, NO HAY FUNCIONES igualmente), el lenguaje definido en esta guía, operaciones booleanas (and, or ,…) y vectores.
- Los procedimientos tendrán variables locales y solamente a través de parámetros (estilo C) se podrán enviar valores a los mismos.
- Cualquier duda por favor consultarla a la lista de la materia: http://groups.google.com/group/compiladores_e_interpretes