~\Desktop\OneDrive\Cuarto\Segundo Cuatri\Compiladores\practica3\notas_p_bison.txt

- hay una parte optativa de la práctica que puntúa por encima de los 1,5 puntos de la práctica.

--- VERSIÓN BÁSICA ---

- habrá un cursor que permitira que el usuario introduzca esos comandos o expresiones.
- hay que hacer un interprete similar a R, Python o Matlab.
- habra que incluir las operaciones arimeticas basicas (con parentesis, simbolos y demas). Por ejemplo
  ```
  > 3 + 2
  (retorno de carro)
  ```
  resulta en:
  ```
  > 5
  ```
- las cadenas deben de contener cadenas válidas en la gramatica del lenguaje y se le asigna un significado de manera que el resultado de este significado se devuelve por pantalla.
- se puede colocar un doble cursor para las soluciones.
- se puede incorporar el concepto de echo: si la linea termina en ; se puede omitir el resultado de la operacion (no se hace echo).
- tambien se pueden añadir variables, como:
  ```
  > a = 1;
  > a = a + 1;
  > b = c + 1;
  ```
  Como la variable c aparece por primera vez a la derecha de una sentencia de asignación, se debe reconocer un error y el interprete debe imprimir un mensaje diciendo que c no está inicializada (no se deberia por defecto inicializar c a 0).
- se puede realizar una declaracion de variables asignando tipo, pero no se aconseja -> es mejor hacerla sin tipos de variable (ni siquiera hacer asignacion dinamica de tipos, siendo el tipo el del valor que asigna en cada momento, no el de la variable).
- se pueden utilizar por defecto valores float, ya definidos de la práctica anterior mediante sus expresiones regulares.
- se pueden adoptar algunos modos de representacion como short, long, etc, aunque luego el almacenamiento se corresponde todos con un float (tampoco tiene demasiada importancia).

- se deben de incorporar funciones sencillas como sin, cos, tan, exp, log... Esto tambien permitira introducir nociones mas complejas de tipos de variables. Por ejemplo, Matlab considera por defecto que toda variable es una matriz. Se pueden extender a arrays o matrices, pero tampoco es lo más interesante. Se podrían incorporar funciones de más de un argumento.
- tenemos que incorporar dos constantes: E y PI, deben estar siempre en el lado derecho de sentencias de asignacion. si aparece en algun momento en el lado izquierdo, el interprete deberia devolver un error.

- se puede extender el conjunto de operadores a aquellos relaciones que permiten discutir el caracter de verdad (>, <, <=...), para enriquecer el conjunto del interprete.

- tambien se podria manejar un fichero de texto con sentencias, de forma que se pueda cargar con una funcion LOAD() e interpretarlo sentencia a sentencia.
- un script de sentencias tambien podria llamarse a si mismo, introduciendo la nocion de recursividad.
- otro comando puede ser QUIT() = devolver el control a aquel script o sentencia que lo llamo o hacer que explote todo y se cierra y sale del interprete.
- comando HELP() con ayuda sobre lo que el interprete permite hacer y como lo permite hacer. tambien podriamos pedir ayuda sobre la ayuda, el interprete en general y ciertos comandos (no es del todo necesario).
- al comienzo de la ejecucion del interprete, debera aparecer una linea diciendo "si quieres ayuda, pulsa ?" o similares.
- comando WORKSPACE() = impresion de variables y valores actuales desde el inicio de la ejecucion del interprete (valor ultimo de esas variables). Si una variable se ha asignado de manera incorrecta, no saldria en el workspace (como b en b = c + 1;).
- comando CLEAR() borrar el WORKSPACE(). no hay variables, despues de clear, workspace está vacio. tambien se puede hacer clear sobre una o mas de una variables especifica (no importa muncho).
- CLEAN() funcion de limpieza de pantalla y se devuelve el cursor a la esquina superior izq de la pantalla.
- conmutar entre las dos opciones de echo con ECHO ON/OFF es util.

- expresiones del tipo 3 + deberian delvolver error si aparece un retorno de carro (no le gusta que aparezca un cursor esperando a que completes, pues esa cadena ya tiene un FIN_CADENA).

- se puede añadir un historico de comandos recuperando comandos previos con los cursores, como en la terminal.
- el objetivo es acercar un interprete de comandos a algo que normalmente utilizamos, pero hay que pensar en el tiempo del que disponemos.

- se pueden incluir max, media... donde el numero de numeros no este definido.
- la version basica tiene que estar muy bien.

- puedo establecer una regla sintactica funcion(), donde funcion esta en la TS y puedo utilizar un puntero a funcion para asociar el comportamiento en C (aparece en el manual de Bison).

- necesario fichero leeme al entregar la practica incluyendo version de flex, bison, linux, gcc e instrucciones para compilar y ejecutar todo. la compilacion se ha de realizar sin warnings ni errores. tambien tendremos el fichero warnings, utilizando -Wall. el texto de ayuda de HELP inicial tambien se puede introducir en el propio fichero LEEME.

- no deberia haber gramaticas ambiguas (errores de flex) al entregar.

- hasta aqui seria la version basica.

-- SOFTWARE PARA LA PRÁCTICA ---

- flex y bison. bison genera analizadores sintacticos para gramaticas LALR. serian gramaticas de la familia gcc y similares. proporcionamos la especficacion de la gramatica. tambien se puede proporcionar un sistema de atributos con semantica. bison puede devolver la gramatica compilada y el funcionamiento de todas las operaciones y funciones que le facilitemos.
- las sentencias bien formadas de la gramatica son cada una de las lineas que podemos escribir desde el cursor hasta el retorno de carro.
- se debe leer el manual de bison.
- para el lexico del interpete de comando escribimos un escript de flex .l como en la P2, y se podrán reutilizar las expresiones regulares para numeros (sin hexadecimales a lo mejor).
- a partir de la lectura de la calc cientifica del manual de Bison salen muchas de las cosas que se están pidiendo para la práctica.
- vamos a aprovechar la TS de las prácticas 1 y 2 para manipular las variables, constantes y funciones.
- como registros de esta TS, han de almacenar cosas distintas (tendran distintos campos).
- la TS se utiliza como una memoria del interpete. WORKSPACE recorre la tabla de simbolos e imprime los valores de las variables almacenadas. CLEAR elimina las variables de la TS, pero deja tal cual las funciones y las constantes.
- campo valor en variables, una funcion tiene comportameinto, argumentos... Podemos utilizar la union de C, definiendo tipos distintos para el mismo espacio de memoria para el contenido del registro. tiene el problema de que se reserva memoria de mas (algo parecido en el manual de bison).
- otra opcion es implementar 3 TS con registros uniformes dentro de cada una de esas tablas de simbolos.

--- VERSIÓN PREMIUM ---

- se puede añadir un script del propio lenguaje con una ejecucion de ejemplo de todas las funcionalidades.
- para que las funciones esten disponibles, se incorporan a la tabla de simbolos al comienzo de ejecucion del interprete, cargando y proporcionando el puntero a la libreria math.h leyendo de un array como con las keywords en la P1.
- tambien se cargan en la TS todos los comandos al principio.
- todo esto son librerias estaticas, luego si yo quiero modificar algo tengo que recompilar el interpete.
- pero sera mejor poder usar sentencias de libreria dinamica como R durante la ejecucion del interprete del tipo IMPORT() (otra funcion que se carga al principio). el argumento de importa es una libreria de la que se cargan en tiempo de ejecucion funciones y constantes.
- para esto, utilizamos dlfcn.h, con 4 funciones basicas:
  - dlopen abre lib
  - dlclose cierra lib
  - dlsym accede a un simbolo
  - dlerror gestiona un error con un simbolo (p. ej. no esta)

  una indicacion es (no importar la libreria, arbrirla, ni intentar acceder a todos los simbolos de lal libreria), la mejor, sera ir a la TS si aparece una funcion
  - si está, se usa
  - si no está, puede pertenecer a alguna de las libs importadas. recorremos la lista de libs con dlsym. si aparece, la incluimos a la TS y ahora se puede usar desde la TS (supone manejar punteros a funciones de la lib).

- para la construccion de las librerias, se deberia llegar a tener un .so.

- una advertencia: si por ejemplo importamos la funcion factorial, deberian funcionar
  tanto factorial(4); como a = factorial(4) + 1;
- las librerias importadas NO se pueden modificar ni eliminar con WORKSPACE.

- puede haber una manera de definir las funciones dentro de un script, como Python
  o R, sobre la sintaxis del propio interprete. Se podrna añadir condiciones o lazos.
  Pero esto ultimo no es una libreria. Esto puede ser muy complicado, y se puede entender
  como código interpetado o compilado en archivo.

--- RESUMEN IMPORTANTE ---

- lo primordial es que la version basica este bien y funcione bien. Luego ya se hará la
  premium. primero hay que tener la version basica bien hecha.
- el consejo es lo antes posible echarle un ojo al manual y a sus ejemplos.