Taller I – Curso 2017 - 2018
Trabajo de Laboratorio

Los equipos de trabajo tienen que ser cooperativos, es decir, todos y cada uno de sus integrantes tendrán que participar para lograr una auténtica integración y cada uno se enriquecerá con la colaboración de los demás.

Planteo del problema

Se desea desarrollar un ***compilador*** para un lenguaje de programación llamado ***CalcuSimple***. Se trata de un nuevo lenguaje de programación que permitirá realizar cálculos aritméticos simples sobre valores numéricos enteros. En este taller vamos a desarrollar un compilador escrito en C++ para dicho lenguaje, que permitirá compilar y ejecutar programas escritos en él.

Todo programa en ***CalcuSimple*** será escrito por el programador utilizando algún editor de texto externo (por ejemplo: `notepad++`). Nuestro compilador tomará el código fuente escrito en dicho archivo y lo analizará sintáctica y semánticamente. En caso de que el programa tenga algún error de sintaxis o de semántica, se informará al programador del primer error encontrado y la compilación finalizará en ese momento. El programador deberá corregir dicho error (modificando el código fuente en el archivo) y luego intentar compilar el programa nuevamente. Cuando no tenga ningún error, la compilación finalizará de manera exitosa y el programa podrá luego ser ejecutado.

Un programa escrito en ***CalcuSimple*** tendrá las siguientes secciones: un encabezado de todo el programa, una sección para declarar variables y otra sección que contendrá las instrucciones del programa. Presentamos a continuación un programa de ejemplo escrito en este lenguaje:

```
PROGRAMA Ejemplo
VARIABLES num x resu
INSTRUCCIONES
LEER x
num = x
MOSTRAR num
resu = DIV 40 num
MOSTRAR resu
```

La primera línea contendrá la palabra reservada `PROGRAMA` seguida del nombre del programa. Podrá haber uno o más espacios en blanco entre la palabra reservada y el nombre del programa. El nombre del programa solamente podrá contener letras y deberá coincidir con el nombre del archivo que contiene el código fuente, el cual tendrá la extensión `.csim`. Recordemos que el programador escribirá el código fuente utilizando algún editor de texto externo (por ejemplo: `notepad++`). Para este ejemplo, el nombre del archivo con el código fuente será `Ejemplo.csim`.

La segunda línea contendrá la palabra reservada `VARIABLES` seguida de los nombres de todas las variables del programa, los cuales sólo podrán contener letras. Podrá haber cero o más variables, pero la palabra `VARIABLES` deberá estar siempre presente. Todas las variables serán siempre de tipo entero en este lenguaje. Por esta razón, **no** se indicará el tipo de las mismas al declararlas. Los nombres de las variables se separarán por uno o más espacios en blanco y se inicializarán automáticamente en cero (**no** habrá variables con valores basura en este lenguaje). Se podrá declarar cualquier cantidad de variables y en cualquier orden, pero **no** podrá haber dos variables con igual nombre. Incluso se podrá declarar variables que no se usen en ninguna instrucción.

La tercera línea contendrá solamente la palabra reservada `INSTRUCCIONES`, que indicará el comienzo de la sección de instrucciones del programa, las cuales se ubicarán a partir de la línea siguiente. Podrá haber cero o más instrucciones, pero la palabra `INSTRUCCIONES` deberá estar siempre presente. Cada instrucción ocupará una línea nueva y habrá tres tipos de instrucciones: la instrucción `LEER` (que leerá por teclado un valor para una variable), la instrucción `MOSTRAR` (que mostrará por pantalla el valor de una variable) y la instrucción de ***asignación***, que copiará el contenido de una variable a otra variable o bien asignará a una variable un valor o el resultado de alguna de las cuatro ***únicas*** funciones predefinidas (`SUM`, `RES`, `MUL`, `DIV`) correspondientes a las cuatro operaciones aritméticas elementales (suma, resta, multiplicación y división). Ninguna instrucción podrá hacer uso de una variable que **no** haya sido declarada en la sección de variables.

En este lenguaje **no** se utilizará ningún separador para las instrucciones. Cada instrucción ocupará una nueva línea, siendo esta la única manera de separar cada instrucción de la siguiente. Todas las instrucciones se escribirán, una por línea, luego de la palabra `INSTRUCCIONES`. La última instrucción ocupará la última línea del archivo fuente. **No** se usarán operadores (`+`, `-`, `*`, `/`, etc.) sino ***únicamente*** las cuatro funciones predefinidas. Tampoco habrá comentarios, estructuras de control (`if`, `while`, `for`, etc.) ni procedimientos o funciones en este lenguaje. Las palabras reservadas `PROGRAMA`, `VARIABLES` e `INSTRUCCIONES`, como las instrucciones `LEER`, `MOSTRAR` y las cuatro funciones predefinidas (`SUM`, `RES`, `MUL`, `DIV`) deberán ir siempre en ***mayúsculas***.

Las instrucciones `LEER` y `MOSTRAR` admitirán como único parámetro el nombre de la variable a cargar o mostrar (según corresponda). **No** se podrá cargar (ni mostrar) más de una variable por vez. La instrucción de asignación tendrá alguna de las siguientes formas: `var = param` o bien `var = FUNC param1 param2` (sobreescribiendo el valor anterior de la variable `var`). En el primer caso, se asignará a `var` el valor de `param`, siendo `param` un valor numérico o bien una variable. En el segundo caso, se asignará a `var` el resultado de aplicar `FUNC` a los parámetros `param1` y `param2`. `FUNC` podrá ser cualquiera de las cuatro funciones predefinidas (`SUM`, `RES`, `MUL`, `DIV`) mientras que tanto `param1` como `param2` podrán ser valores numéricos o bien variables.

En cualquiera de los tres tipos de instrucción (`LEER`, `MOSTRAR` o ***asignación***) podrá haber uno o más espacios en blanco separando cualquiera de los elementos involucrados. Por ejemplo, entre la instrucción `LEER` y la variable a cargar, o también entre el operador de asignación y el nombre de la función a invocar, o también entre los dos parámetros de la función invocada, etc. Incluso podrá haber cero o más espacios en blanco al comienzo y al final de cada línea dentro del archivo fuente.

Se muestra a continuación una ejecución de ejemplo para el programa anterior:

```
Ingrese valor para x: 5
num vale: 5
resu vale: 8
```

La instrucción `LEER` le mostrará al usuario el mensaje `Ingrese valor para` seguido del nombre de la variable a cargar. El usuario ingresará el valor para la variable y pulsará enter para seguir con la ejecución. En caso de que el usuario ingrese algo que **no** sea un número entero (por ejemplo, una secuencia de letras o un número real) se mostrará un mensaje de error apropiado y la ejecución finalizará en ese momento. La instrucción `MOSTRAR` le mostrará al usuario el mensaje `var vale` (siendo `var` el nombre de la variable a mostrar) seguido de su valor. En caso de que el programa llame a la función `DIV` (como ocurre en el programa de ejemplo), al momento de ejecutarla se deberá controlar que el segundo parámetro **no** valga cero. En caso de que valga cero, se mostrará otro mensaje de error apropiado y la ejecución finalizará en ese momento.

Para utilizar el compilador, el usuario digitará tres comandos por teclado, que le indicarán al compilador las diferentes acciones a realizar. Los tres comandos son:

- `compilar`: compila un programa determinado, dado su nombre.
- `ejecutar`: ejecuta un programa determinado, dado su nombre.
- `salir`: finaliza la sesión de trabajo con el compilador.

Se muestra a continuación una sesión de trabajo con el compilador, para familiarizarse con el uso de los tres comandos anteriores. En cada paso de la sesión, el compilador solicitará un comando. El usuario podrá digitar cualquiera de los tres comandos disponibles:

```
Ingrese comando: compilar Ejemplo
Ejemplo compilado exitosamente
Ingrese comando: ejecutar Ejemplo
Ingrese valor para x: 5
num vale: 5
resu vale: 8
Ingrese comando: salir
Hasta la próxima
```

Cuando el usuario digite un comando y presione enter, lo primero que hará el compilador será chequear la sintaxis del comando ingresado. Verificará que efectivamente se trate de alguno de los tres comandos válidos y que, para los comandos `compilar` y `ejecutar`, lo digitado enseguida efectivamente sea un nombre de programa válido (solamente contenga letras y esté formado por una única palabra). Podrá haber uno o más espacios en blanco entre el comando y el nombre del programa. Para el comando `salir`, verificará que no haya nada más escrito a continuación del comando. Podrá haber cero o más espacios en blanco al comienzo y/o final de la línea. En caso de haber algún error de sintaxis en el comando ingresado, se emitirá al usuario un mensaje de error apropiado y el comando **no** será ejecutado, pidiéndole a continuación que ingrese nuevamente un comando. En caso de que no haya ningún error de sintaxis, se procederá a la ejecución del mismo.

En caso de tratarse del comando `compilar`, se mostrará un mensaje de éxito siempre que la compilación sea exitosa (como en la sesión de ejemplo) o bien un mensaje de error indicando el primer error detectado en la compilación del programa (en cuyo caso la compilación finalizará en ese momento). Si se produce un error, puede a su vez ser por diferentes motivos. Por ejemplo, pretender compilar un programa que no existe o bien un error de sintaxis o de semántica en el programa que se está compilando (declarar una variable cuyo nombre no está formado solo por letras, pretender usar una variable que no fue declarada, pasarle una cantidad incorrecta de parámetros a cualquiera de las cuatro funciones predefinidas, etc.). Tras la compilación del programa (tanto si es exitosa como si presenta error de compilación), se retornará a la sesión de trabajo con el compilador, y se le volverá a pedir al usuario que ingrese un nuevo comando.

En caso de tratarse del comando `ejecutar`, se procederá a ejecutar el programa indicado, lo cual se hará dentro de la misma sesión (como en la sesión de ejemplo) o bien se emitirá un mensaje de error (en cuyo caso la ejecución del programa indicado **no** se llevará a cabo). Si se produce un error, puede a su vez ser por diferentes motivos, como pretender ejecutar un programa que no existe, o que existe pero no compiló exitosamente. Si no hay ningún error, se procederá a la ejecución del programa indicado (la cual, a su vez, puede ser exitosa o finalizar debido a un error de ejecución como, por ejemplo, intentar hacer una división entre cero). Tras la ejecución del programa (tanto si es exitosa como si presenta error en su ejecución), se retornará a la sesión de trabajo con el compilador, y se le volverá a pedir al usuario que ingrese un nuevo comando.

Dentro de una misma sesión de trabajo, el usuario podrá compilar y ejecutar tantos programas como desee (en la sesión de ejemplo se compiló y ejecutó solamente un programa) y hacerlo en el orden que prefiera. Por ejemplo, compilar varios programas y ejecutarlos luego de compilarlos a todos, o bien compilar un programa y ejecutarlo inmediatamente, y luego de eso compilar y ejecutar un segundo programa, etc. Incluso podrá compilar y/o ejecutar varias veces un mismo programa dentro de la misma sesión. La sesión finalizará cuando se ingrese el comando `salir`.

Se deberá realizar un análisis detallado de cada comando y controlar <u>todos</u> los errores que se puedan producir, tanto a nivel de sintaxis y de semántica como a nivel de la ejecución. Se valorará positivamente la emisión de mensajes de error al usuario que sean lo más específicos posible.

Aproximación a la solución

Para la implementación del compilador se deberá utilizar, como mínimo, las siguientes estructuras de datos (también se podrá definir otras estructuras de datos adicionales, si resultan de utilidad):

Las variables de un programa escrito en ***CalcuSimple*** se almacenarán en un ***ABB* (*árbol binario de búsqueda*)** en C++, ordenadas por nombre de variable. Junto a cada variable, se almacenará su valor. Recordar que el valor inicial de toda variable siempre será cero. Al momento de compilar la sección de variables del programa, se cargarán en el árbol todas las variables declaradas en ella. Cada variable ocupará un nodo del árbol, acompañada de su valor. En caso de producirse algún error de compilación en dicha sección, el proceso se cancelará en ese instante y el árbol deberá ser vaciado, ***liberando*** toda la memoria dinámica correspondiente al mismo.

Las instrucciones de un programa escrito en ***CalcuSimple*** se almacenarán en una ***Lista*** en C++, en el mismo orden en que se encuentran en el archivo fuente del programa. A medida que se vaya compilando cada instrucción del programa, sus datos serán insertados en un nuevo nodo en la lista, a continuación de la última instrucción insertada. Se deberá analizar en detalle qué datos de cada instrucción corresponde almacenar en cada nodo de la lista. Recordar que habrá tres tipos de instrucción posible (lectura por teclado, escritura por pantalla y asignación, la cual, a su vez, puede llamar a cualquiera de las cuatro funciones aritméticas predefinidas). En caso de producirse algún error de compilación en una instrucción, el proceso se cancelará en ese instante y la lista deberá ser vaciada, ***liberando*** toda la memoria dinámica correspondiente a la misma.

Al compilar un programa escrito en ***CalcuSimple***, se deberá ir leyendo, una por una, las líneas del archivo conteniendo el código fuente del programa (archivo `.csim`). Cada línea del archivo será cargada en un ***string dinámico***. Para este taller, asumiremos que el tamaño máximo del string (dado por la constante `MAX` de la definición del tipo `string`) será suficiente para contener todos los caracteres que hay en la línea. Dado que el archivo fuente (archivo `.csim`) es un archivo de ***texto***, cada uno de los caracteres de la línea deberá leerse mediante la instrucción `fscanf` de la librería `<stdio.h>` del siguiente modo: `fscanf (file, "%c", &car);` siendo `file` el puntero a `FILE` obtenido al abrir el archivo (se deberá abrir en modo `"r"`) y `car` una variable de tipo `char`. Cada línea del archivo finalizará con enter. Tomando en cuenta esto, se deberá implementar un procedimiento que cargue una línea del archivo a un string dinámico en memoria.

Una vez que el string dinámico haya sido cargado a memoria, se deberá realizar un procesamiento del string (habitualmente se llama ***parsing*** a dicho procesamiento) para reconocer su contenido. Por ejemplo, la primera línea del archivo debería ser de la forma `"PROGRAMA nombre"` (pudiendo haber varios espacios en blanco entre la palabra `PROGRAMA` y el nombre). Será necesario partir el string en substrings, a efectos de reconocer el contenido de cada uno de ellos. Una vez que el string haya sido parseado, se deberá tomar las acciones que correspondan para continuar con el proceso de compilación (o detenerlo, en caso de que haya algún error). En cuanto a la lectura por teclado de cada comando (`compilar`, `ejecutar`, `salir`), se procederá de igual manera: la línea digitada por el usuario será cargada en un string y se aplicará un proceso de ***parsing*** al mismo. La única diferencia es que, en este caso, el string será leído por teclado en vez de desde archivo.

En caso de que la compilación sea exitosa, se tendrá en memoria un ***ABB*** con las variables del programa y una ***Lista*** con los datos de las instrucciones del programa. En ese momento, el contenido del ABB será bajado a un archivo ***binario*** con extensión `.vars` y el contenido de la lista será bajado a otro archivo ***binario*** con extensión `.inst`. Dichos archivos **no** serán de texto, sino de ***bytes***, por lo que la escritura en ellos se deberá realizar mediante la instrucción `fwrite`. Los nombres de ambos archivos deberán coincidir con el nombre del archivo fuente del programa. Por ejemplo, si el archivo fuente es `Ejemplo.csim`, los dos archivos resultantes de la compilación se llamarán `Ejemplo.vars` y `Ejemplo.inst`. Una vez generados ambos archivos, tanto el ABB de variables como la Lista de instrucciones deberán ser vaciados, ***liberando*** toda la memoria dinámica correspondiente a los mismos. La existencia de los archivos `.vars` y `.inst` en disco indicarán que el programa compiló exitosamente y que luego podrá ser ejecutado.

Al momento de ejecutar un programa escrito en ***CalcuSimple***, lo primero que se deberá verificar es que haya sido compilado exitosamente. En caso de que los archivos `.vars` y `.inst` existan en disco, significa que el programa compiló exitosamente y puede ser ejecutado. Para la ejecución, se cargarán nuevamente a memoria el ABB de variables respaldado en el archivo `.vars` y la Lista de instrucciones respaldada en el archivo `.inst`. Dado que ambos archivos son ***binarios***, la lectura de los datos en ellos se deberá realizar mediante la instrucción `fread`. Para la ejecución, se deberá ir ejecutando, una a una, las instrucciones contenidas en la Lista de instrucciones (en el mismo orden en que aparecen en ella). Puede llegar a pasar que, durante la ejecución, ocurra un error en alguna de las instrucciones (por ejemplo, una división entre cero) en cuyo caso, la ejecución de todo el programa finalizará en ese momento. En otro caso, se ejecutarán con éxito todas las instrucciones contenidas en la Lista. Finalizada la ejecución (tanto si es exitosa como si se produce un error), tanto el ABB de variables como la Lista de instrucciones deberán ser nuevamente vaciados, ***liberando*** toda la memoria dinámica correspondiente a los mismos. La diferencia con el proceso de compilación es que, en este caso, **no** serán bajados a disco de nuevo.

Primera Entrega - *Análisis y Diseño*

Fecha de entrega: **Lunes 19 o Martes 20 de Febrero de 2018** (según el día en que toque la tutoría, en horario de clase). Se debe entregar un documento **impreso** (debidamente organizado) conteniendo los siguientes elementos:

1. Definición de todas las estructuras de datos necesarias para representar el problema, y escritura en C++ de todos los tipos de datos correspondientes.
2. Diagrama de módulos conteniendo los módulos identificados hasta el fin de la semana 2 junto con las inclusiones correspondientes.
3. Seudocódigo de la ejecución de cada comando. El seudocódigo correspondiente a cada comando debe ser lo más ***detallado*** posible, a efectos de permitir luego definir, a partir de él, todos los procedimientos y funciones que sean necesarios para su resolución.
4. Cabezales sintácticos de ***todos*** los procedimientos y funciones a incorporar en cada uno de los módulos, los cuales surgirán a partir del seudocódigo realizado para cada comando.
5. Cronograma de planificación de actividades del grupo indicando para cada tarea a realizar una estimación de fecha de inicio y fecha de fin.
6. Explicación de todas las decisiones de diseño que haya ido tomando el equipo al realizar el diseño de la solución.

Segunda Entrega - *Implementación*

Fecha de entrega: **Martes 13 de Marzo de 2018** via e-mail a la dirección del docente tutor, antes de las 24:00 hs. Se debe entregar un archivo comprimido (en formato zip) conteniendo el código fuente de la aplicación en C++, el proyecto de CodeBlocks y ***excluyendo*** el código objeto (archivos binarios (`.o`) y archivo ejecutable (`.exe`) de la aplicación).

Dicho archivo comprimido debe contener también un archivo de texto llamado `Integrantes.txt` con nombres, apellidos y números de cédula de los integrantes del equipo. El nombre del archivo comprimido debe llamarse: `GrupoXX.zip` donde XX será el número de grupo que se le asignará a cada grupo tras la presentación del taller.

Metodología de Trabajo

Se deberá trabajar en forma ordenada, elaborando el documento solicitado para la primera entrega y haciendo un programa de prueba por cada módulo implementado. Además de los ítems especificados anteriormente, el grupo también deberá entregar cualquier otra documentación que el tutor considere oportuna durante el desarrollo de las distintas etapas del taller.