Trabajo practico nº 3

Interprete de SCEQL

75.40 Algoritmos y Programación I

Cátedra: Wachenchauzer

Integrantes: Arturo Alejandro Yep Vela – Alexis Albanese

Ayudante a cargo: Rodrigo Velez

Practica: Barbara

Fecha de Entrega: 17/11/2017

Nota:

Etapa 1: Planteamientos del Programa

Se comenzó por plantear que se requería que hiciera el programa. Esto fue que interpretara archivos en forma **SCEQL**.

SCEQL es un lenguaje de programación minimalista que solo acepta 7 caracteres con una función especifica cada uno.

Carácter	Función
\	Si el byte al frente es cero, "saltar" hasta el comando siguiente al / correspondiente
/	"saltar" hacia atrás, hasta el comando \ correspondiente
*	Desencolar el byte al frente, imprimirlo interpretado como un caracter ASCII, y luego encolarlo
=	Desencolar un byte y encolarlo
-	Decrementar el byte que está al frente de la cola (sin moverlo de lugar)
_	Incrementar el byte que está al frente de la cola (sin moverlo de lugar)
!	encolar un 0

Como se puede observar, hay dos caracteres que tienen relacion entre si, “\” y “/”. El codigo entre barras es repetido **hasta que el byte de al frente es cero**, y deben venir de a pares.

El programa debe aceptar el nombre del archivo **.sceql** por parámetro al ser ejecutado en la consola. También un parámetro opcional es el modo **DEBUG** “-d” o “-debug”, esto permite que el usuario vea el código fuente y maneje el mismo, presionando cualquier tecla, el avance de la ejecución.

Etapa 2: Diseño del Programa

Una vez que se planteo que debería hacer el programa, se continuo con diseñar como debería hacerlo.

1. Tomar el argumento necesario del archivo **.sceql**, y el modo debug solo si se requiere.

2. Abrir el archivo y leer su contenido
3. Guardar el contenido del archivo en una cadena

4. Reconocer la posiciones de las barras invertidas y sus barras correspondientes

5. Recorrer el código y ejecutarlo.

6. De ser activado el modo debug, la ejecución del código debe continuar cuando el usuario apriete una tecla y mostrar su código.

7. Ignorar cualquier carácter fuera de los 7 aceptados

Etapa 3: Desarrollo

Para cumplir el 1er paso, se requirió el uso del modulo argsparse, que hace posible, de una manera sencilla, que el usuario ingrese parámetros, necesarios u opcionales.

Se lee el archivo ingresado, guardando el contenido en una variable en forma de cadena.

Cuando el archivo se termina de recorrer, se pasa a ejecutarlo, pero antes se necesita encontrar las posiciones de las barras invertidas y sus barras correspondientes.

Una vez obtenidas, se continua con el proceso de ejecución del interprete de SCEQL.

En este paso se verifica si el modo debug fue o no fue llamado. De ser asi, se imprime en pantalla el mensaje paso a paso y un segmento de 100 caracteres del código fuente SCEQL en ejecución, indicando cual es el carácter a evaluar.

De ser llamado también se detiene la ejecución automática, esperando que el usuario presione una tecla para que continué al paso siguiente.

Se continua evaluando el carácter actual, dependiendo de cual sea se procede a efectuar la funcion a la cual esta destinada.

Etapa 4: Funcionamiento del Programa

Antes de ejecutar el programa se deben ingresar sus parámetros, estos son: el nombre del archivo .sceql, el cual es necesario y el parámetro opcional “-d”, o “-debug”, el cual habilita el modo debug.

Los parámetros son “tomados” por el uso del modulo argsparse, quien puede tomar parámetros de la linea de ejecución en la consola y contar con esos datos dentro del código.
De no haber ingresado el parámetro del nombre del archivo, el programa no podrá iniciar.

Ejecutado el programa, se procede a abrir y leer el contenido del archivo, de no ser posible abrir el archivo, se levantara un error que sera atrapado, imprimirá en pantalla la razón y se terminara su ejecución.

Para leer y guardar el contenido del archivo, se procedió a guardar las lineas del mismo en una lista vacía, para luego juntar usando **join.** Se eligió este procedimiento por ser mas eficiente que concatenar cadenas.

Obtenido el contenido del archivo, se procede a llamar la función **ejecutar**, quien se encarga de ejecutar el código **.sceql** .

Antes de comenzar a ejecutarlo, se realiza una pre-lectura del código, llamando a la función **obt_pos_barras**, quien se encarga de recorrer el código y de devolver dos diccionarios, uno usando claves como posiciones de barras invertidas y valores como las posiciones de las barras correspondientes, y el segundo contiene lo mismo, pero las claves y valores invertidos.

Para obtener la relación de correspondencia entre barras, se hizo uso del **TAD Pila**, que tiene la característica especial que lo ultimo ingresado es lo primero en salir (F.I.L.O.) . Apilando las posiciones de las barras invertidas, al encontrar una barra, es la correspondiente a la ultima apilada, se desapila y se procede a ingresar los datos en los diccionarios.

Si durante este proceso, se intenta desapilar y la pila esta vacía, esto significa que las barras no se encuentran a pares, se levanta un error, y se atrapa en la función **main**, imprimiendo en pantalla el motivo y terminando su ejecución

Lo mismo sucede si al terminar de recorrer el código la pila sigue teniendo elementos, se continua con el mismo procedimiento.

De ser que la función **obt_pos_barras**, termine sin levantar errores se devuelven los dos diccionarios y se continua el funcionamiento de la función **ejecutar**.

En esta función se utiliza el **TAD Cola**, tiene la característica de que lo primero en entrar, es lo primero en salir (F.I.F.O.).

para poder usarla, se agrego un método mas a la clase Cola, ademas de los normales (esta_vacía, encolar, desencolar), este método fue **cambiar_primero**, que permite cambiar el primer elemento de la fila sin que se pierda el orden.

Continua con un ciclo indefinido **while** mientras el contador de posiciones sea menor a la cantidad de elementos en el código. Una vez comenzado, se pregunta si esta seleccionado el modo debug, de ser así, se llama al función **debug**.

La función se encarga de imprimir en pantalla el mensaje impreso hasta el momento, el estado acual de la cola, para lo que se le agrego un metodo **imprimir_cola,** y un fragmento de 100 caracteres del código en ejecución, indicando el carácter a evaluar. El fragmento va cambiando dependiendo de la posición del carácter.

También, la ejecución del código se vuelve manual, esperando que el usuario presione una tecla para seguir al paso siguiente.

De no estar seleccionado el modo debug, se continua con la ejecución automática del código, una vez finalizado se termina la ejecución del programa.

Para la ejecución del código, se hizo uso de un **diccionario de funciones**, donde las claves son los caracteres del lenguaje a interpretar y los valores son las funciones que se llaman dependiendo del caso. A modo de generalizar el uso de las funciones, a todas se les pasaron los mismos parámetros, del mismo modo que todas devuelven los dos resultados, dependiendo de cual cambien.

Se decidió importar esta parte del código ya que las funciones se consideran de uso privado.

Estas fueron 4 funciones:

- _barras : se llama cuando el elemento a considerar es “\” o “/”, trabaja con los diccionarios **camino_1** y **camino_2** y la cola actual, dependiendo de si el carácter actual es 0, “\” salta a su “/” correspondiente, o sigue avanzando normal. En el caso de “/”, “salta hacia atrás” a su “\” correspondiente.

- _igual_admiración : esta función se encarga de los elementos “!”, “=”. Modifica la cola según el elemento a evaluar. En ambos caso avanza a la siguiente posición.

- _guiones : los elementos a que llaman a esta función son “-” y “_”. Aumentan o decrementan el primer valor de la cola sin cambiar el orden. En ambos caso avanza a la siguiente posición.

- _asterisco : Es la uncia función que tiene un solo elemento que la llama “*”. desencola el primer elemento de la cola, lo traduce según la tabla **ASCII**, lo

agrega a **mensaje**, y lo imprime. Avanza a la siguiente posición y es la única función que devuelve **mensaje**, modificado.