Y Compiler

Como correr:

Y Compiler es una aplicacion web implementada completamente del lado del cliente, empleando JavaScript (con ayuda de Bootstrap, JQuery y Jison). Por este motivo, no es necesario ningun tipo especial de procedimiento para correr el compilador, puede ser abierto desde cualquier navegador.

Entorno de desarrollo:

El compilador fue desarrollado sobre Linux empleando FireFox como navegador por defecto.

Funcionamiento

El funcionamiento del compilador esta dividido en varias fases para ayudar al usuario a estar conciente de todas las fases de un compilador y poder observar con detalle como estas se llevan a cabo y de los distintos tipos de errores que pueden ocurrir durante cualquiera de estas fases.

El orden de eventos normal para un proceso de compilacion exitoso es el siguiente:

1. Obtener el codigo fuente.

   Este paso puede ser conseguido de dos formas: subiendo un archivo de texto a la aplicacion o escribiendo el nombre de un nuevo archivo, seleccionando la opcion “nuevo” y escribiendo a mano el codigo fuente. Si el texto es subido de un archivo de texto, este es completamente editable en cualquier momento.

2. Seleccionar el boton “compilar”

   Al hacer click sobre este boton dara comienzo el proceso de compilacion. La primer etapa es el analisis lexico, luego se realiza el analisis sintactico, luego se construye el AST (Abstract Syntax Tree) , y finalmente se emplea este arbol para extraer toda la informacion util de la entrada, se procesa y compila. Como resultado se obtiene:

   1. Tabla de Simbolos

      Una tabla empleada por el compilador en la siguiente fase que contiene toda la informacion necesaria sobre todas las funciones/metodos/constructores/sub-bloques, etc (Scopes para abreviar), detalla todos los Scopes, sus tamanios y el offset de cada variable dentro del sistema y otros detalles importantes para la compilacion. Las clases no se listan aqui.

   2. Lista de Clases

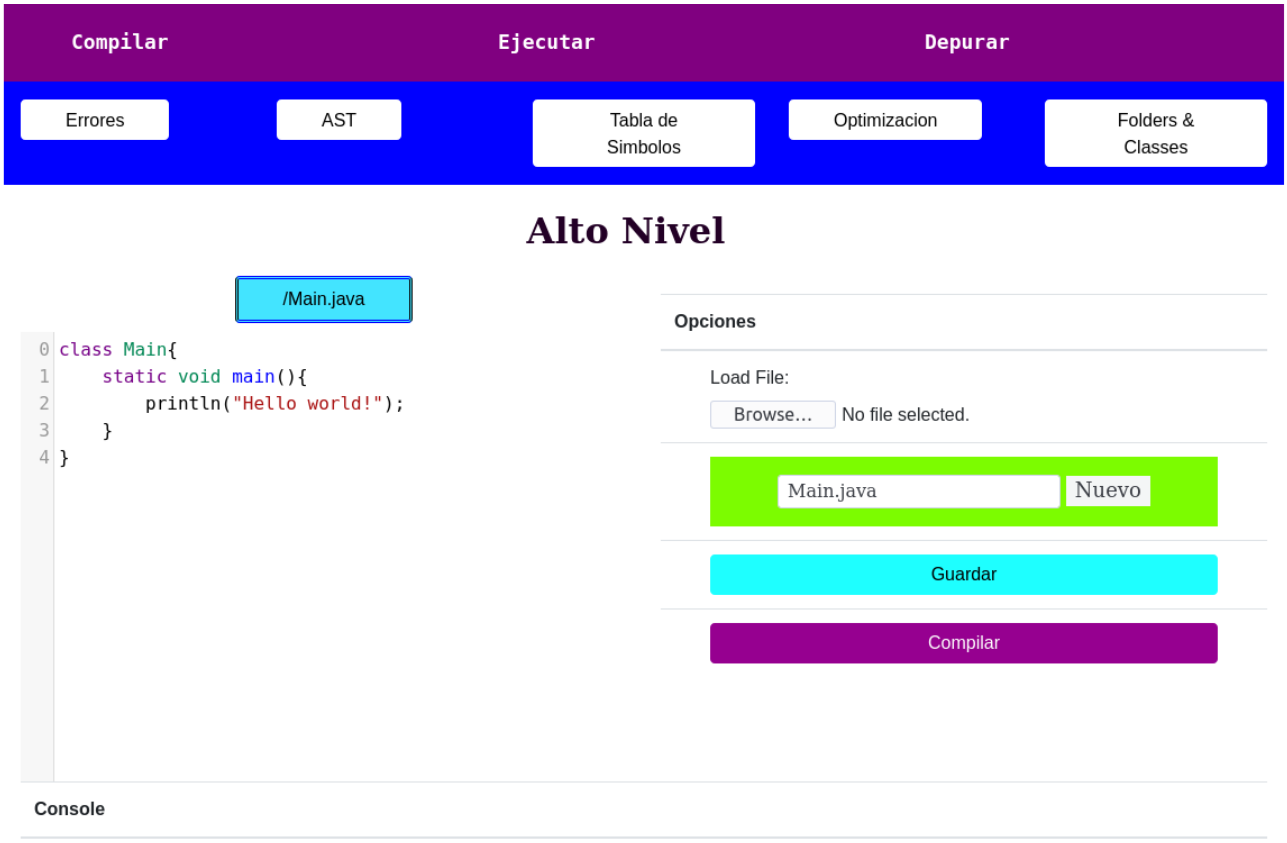
      Las clases se compilan y almacenan en una estructura diferente de la Tabla de Simbolos. Esto porque todos los campos de una clase son relativos a esa instancia de clase, no aun scope determinado (como es el caso de las variables locales) Al finalizar la compilacion se debe mostrar una lista de todas las clases ingresadas en el sistema junto con toda la

informacion pertinente a estas, incluyendo las clases nativas (Object y String) asi como toda la informacion obtenida de un proceso de heredacion.

3. AST

   El AST (Abstract Syntax Tree) es una representacion en forma de Arbol de la sintaxis de la entrada, es util para procesar toda la informacion contenida en esta.

Todas y cada una de estas 3 estructuras es mostrada al usuario si la compilacion fue exitosa, cada una se encuentra ubicada en su propia tabla y se puede navegar de una a otra con total libertad. Tras compilar todas estas estructuras comienza la siguiente fase de la compilacion: La generacion de Codigo Intermedio. El codigo intermedio empleado por este compilador es Codigo 3 Direcciones. (Llamado asi porque solo se puede referenciar a 3 posiciones de memoria al mismo tiempo). Si ocurrio algun error durante esta etapa se le notificara del error y no se le permitira continuar a la siguiente fase. Sin embargo, si el proceso es exitoso entonces el nombre del boton cambia: “Finish compilation” . Y el usuario podra seleccionar el boton para continuar a la siguiente fase. Se adjunta un screenshot del entorno principal del compilador:



Si la primera fase es exitosa el boton morado que dice “Compilar” cambiara el texto a “Finish compilation” y el usuario podra continuar al a siguiente fase:

Generacion de Codigo Intermedio

La generacion de codigo intermedio toma lugar luego de haber compilado exitosamente 3 estructuras importantes: El AST, la Tabla de Simbolos y la lista de Clases.

Empleando estas 3 estructuras, se vuelve a recorrer el AST, pero esta vez se emplea la informacion extraida durante la primera fase para poder generar el codigo intermedio. El codigo intermedio es una representacion del sistema escrito en lenguaje de alto nivel en codigo de bajo nivel en el que cambia totalmente el paradigma y no existen muchos de los conceptos que conocemos en alto nivel. Este codigo puede ser empleado posteriormente para:

Traducir a codigo maquina :

Dependiente del procesador de la computadora objetivo asi como el sistema operativo que esta emplea. En teoria puede ser traducido directamente desde el codigo intermedio al codigo maquina.

Traducir a codigo Assembler:

El codigo Assembler es muy similar al codigo intermedio, sin embargo existen diferentes tipos de lenguaje Assembler, basados en el procesador objetivo. El codigo assembler puede ser compilado para generar un archivo ejecutable que contiene la representacion en codigo maquina del sistema compilado. Al compilador de algun lenguaje Assembler se le conoce como Ensamblador y hoy en dia es casi imposible conseguir un Ensamblador que tenga como objetivo un procesador actual.

Ejecutar el codigo intermedio:

El codigo intermedio puede ser ejecutado directamente por un interprete y saltarse el paso de la compilacion final a codigo maquina para que lo ejecute el procesador. Es precisamente esta la funcion que realiza el Y Compiler, tras finalizar la generacion de codigo intermedio se puede emplear esta misma aplicacion (en una tab aparte) Para ejecutar el codigo generado. Esto es a su vez independiente del computador y el procesador donde se esta ejecutando.

Si la generacion del codigo intermedio es exitosa, sera redirigido automaticamente a la tab de ejecucion. En el text area de la izquierda se puede observar el codigo 3D generado, y a la derecha se puede observar la consola. Sobre la consola se encuentra el boton para comenzar la ejecucion del codigo. Se adjunta un screenshot ilustrativo de esta fase:

Consideraciones:

- Si ocurre un error durante la fase de genearcion de codigo sera redireccionado a la tab de ejecucion, sin embargo, se le notificara del error. Los detalles del/los error(es) esta contenida en una tab aparte nombrada Errores.
- Si realiza un cambio al codigo fuente tras haber finalizado la primer etapa, pero antes de comenzar la segunda y selecciona “Finish compilation” el cambio efectuado no se vera reflejado. Para aplicar el cambio debera resetear el ciclo de compilacion de regreso al principio, puede hacer esto seleccionando el titulo que dice “Alto Nivel” Si lo hace, observara como cambia el texto del boton de regreso a la opcion de compilar.
- Para generar codigo intermedio de manera exitosa se debe saber donde comienza la ejecucion del programa. Por defecto, este buscara la clase Main y posteriormente el metodo: static void main(). Si alguno de estos no se encuentra en el programa fuente se informara al respecto y no sera posible continuar a la siguiente fase. Sin embargo, el suario es capaz de seleccionar cualquier metodo estatico valido como punto de inicio de ejecucion. Para seleccionar un metodo distintinto al default como punto de entrada, simplemente dirijase a la tabla de clases y seleccione cualquier metodo estatico que NO tome ningun parametro. Al hacerlo, este metodo se coloreara verde, podra regresar a la tab de alto nivel y generar codigo intermedio empleando el metodo seleccionado como punto de entrada. Se adjunta un screenshot sobre esta opcion:

Name	Parent	visibility	Abstract	Final	CC	ID	File
Main	Object	public	false	false	5	2	/ArrayNativesTest(1)(1)(1).java
Category	**visibility**	**static**	**Name**	**Type**	**Inherited**	**Abstract**	**final**
method	public	false	equals-Object	boolean	true	false	false
method	public	false	getClass	String	true	false	false
method	public	true	printArray-array|int-String	void	false	false	false
method	public	true	backToString-array|char	String	false	false	false
method	public	true	StringToDouble-String	double	false	false	false
method	public	true	myCustomEntryPoint	void	false	false	false

Tras haber seleccionado un punto de comienzo valido, y si no hay errores el compilador generara el codgio intermedio correspondiente al programa y sera redirigido a la seccion de ejecucion.

En este punto puede considerarse la compilacion como terminada y exitosa. Para poder emplear el codigo intermedio generado, puede hacer uso del interprete de 3D adjunto en esta misma aplicacion.

El codigo 3D a ejecutar por el interprete puede ser el generado por el compilador o puede escribir manualmente codigo 3D en el textarea y observar el resultado de su ejecucion.

En este punto puede optar por diferentes acciones:

- Ejecutar el codigo intermedio

  Esta opcion toma la entrada en el textarea (generalmente llenada con la salida del compilador) y la ejecuta. La salida se puede observar en la consola, ubicada en esta misma tab.

- Depurar el codigo intermedio

  Otra opcion es depurar el codigo intermedio, esta opcion le permitira designar breakpoints en cualquier linea del codigo y seleccionar las opciones comunes de un debugger:

  i. Continuar ejecucion

     Continua la ejecucion desde el punto en que se encuentra y no se detiene hasta finalizar.

  ii. Siguiente linea

     Ejecuta unicamente la siguiente instruccion.

  iii. Siguiente breakpoint

     Ejecuta todas las instrucciones siguientes hasta encontrar un punto breakpoint.

  iv. Saltar

     Ejecuta unicamente la siguiente instrucion, a menos que la instruccion sea un call. En ese caso ejecuta todas las instrucciones pertenecientes al call y se detiene tras regresar del call.

v. Detener

   Resetea la ejecucion al principio.

Todos estos metodos le permiten al usuario visualizar todas las estructuras empleadas por el interprete.

Sin embargo, el Heap muestra unicamente un maximo de 2000 celdas. Si el programa utiliza mas de esta cantidad se seguira ejecutando pero no se mostrara en pantalla. Lo mismo aplica para el Stack, pero el Stack tiene un limite menor: 1000 celdas en display. Las estructuras reales continuan creciendo pero no se muestran al usuario.