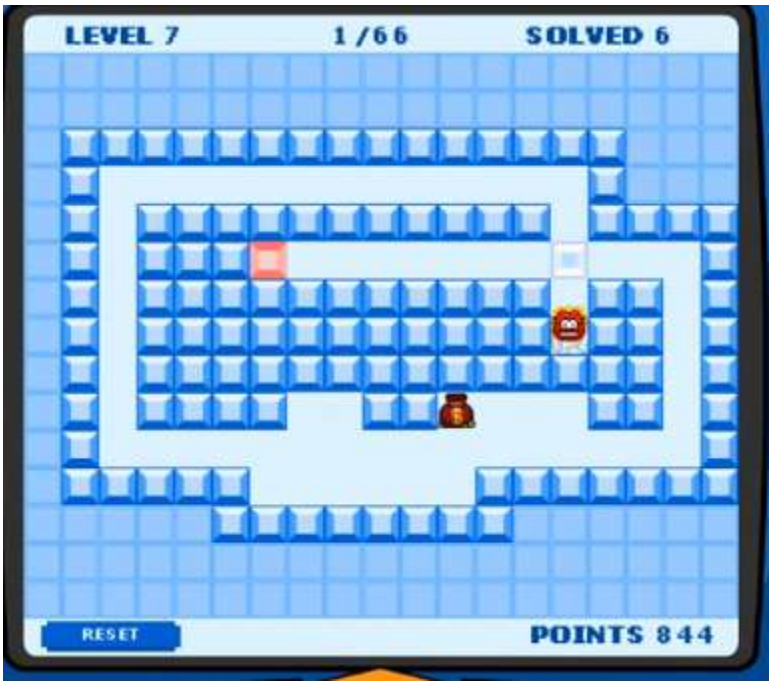
Proyecto Compiladores e Intérpretes



http://www.clubpenguin.com/help/help-topics/mini-games/thin-ice

http://www.youtube.com/watch?v=YWB0CV6WBVg

Se desea construir un video juego denominado Thin Ice, con la diferencia de que el personaje principal en lugar de ser controlado por una persona será controlado por un lenguaje de programación que será posteriormente interpretado y ejecutado paso a paso con cierto intervalo de tiempo de espera entre cada ejecución de una instrucción.

Este lenguaje de programación deberá contener diversos tipos de instrucciones como lo son:

1. **Cuerpo programa:** todos los programas comenzaran por la palabra *programa()* seguida de los operadores de inicio y cierre de bloque que son definidos por las llaves { = inicio de bloque, } = fin de bloque. Por ejemplo:

   ```
   programa()
   {
   }
   ```

2. **Instrucciones de control del personaje:** Son aquellas que controlan el movimiento del personaje y permiten su direccionamiento y control direccional, entre estas tenemos:
   2.1. *avanzar(entero cantidadCuadros),* avanza en la dirección que se encuentre observando el personaje la cantidad de espacios especificados en el parámetro de cantidad de cuadros,

si consigue un obstáculo en la dirección en la cual se encuentra avanzando debe detenerse.

Ejemplo: avanzar(2); //avanza al personaje 2 cuadros si no hay un obstáculo
avanzar(2); //avanza al personaje 1 cuadro si existe un obstáculo en la posición 2

2.2. *mirarArriba(), mirarAbajo(), mirarIzquierda(), mirarDerecha(),* permite direccionar el personaje del juego, en la dirección planteada sin avanzar ninguna posición.

3. **Tipos de datos:** Solo se permite la declaración de variables simples y vectores en tipo de dato entero y lógico (boolean).

4. **Definición de tipos de bloque:** como se definió inicialmente los programas posee las llaves como indicadores de inicio y fin de bloque, pero esa definición de bloque se extiende a otras instrucciones que se encuentren conformadas por más de una instrucción en su interior para evitar la ambigüedad en su inicio y fin como por ejemplo en los si (if) en los ciclos (para), etc. En palabras más simples las llaves marcan claramente el inicio y fin de una lista de sentencias.

5. **Instrucciones de control:** Son aquellas que permiten controlar el flujo de los programas planteados entre estas tenemos:
   5.1. *Ciclo para:* posee la estructura *para(inicialización ; paso ; verificación) {…}* donde cada uno de estos es una expresión, en el caso de inicialización y paso son expresiones de cualquier tipo aunque la lógica indica que inicialización y paso deben ser asignaciones, y verificación una operación lógica, pero esto debería ser comprobado a nivel semántico en lugar de tratar de expresarlo a nivel sintáctico.
   5.2. *Ciclo repita hasta que:* posee la estructura clásica del mismo *repita ... hasta* que condición; y no requiere del uso de sentencias de inicio y fin de bloque debido a que esto ya se deja claro con la estructura de la instrucción.

6. **Expresiones Lógicas, Aritméticas y Relacionales:**
   6.1. Están permitidas las expresiones de asignación (:=) y los operadores matemáticos de suma (+), resta (-), multiplicación (*), módulo (mod) y división (/).
   6.2. Las operaciones aritméticas producen como resultado un valor entero.
   6.3. Las operaciones relacionales son: menor (<), menor o igual (<=), mayor (>), mayor o igual (>=), igual (=) y diferente (<>).
   6.4. Los operadores lógicos son los de conjunción lógica (Y), disyunción lógica (O) y Negación Lógica (!).
   6.5. Las operaciones lógicas y relacionales producen como resultado un valor booleano.
   6.6. Las estructuras condicionales se reducen a condicionales simples si(condicion){…} sino{…} “if then else” simples o anidados. Teniendo la parte del sino como opcional.

7. Vectores: El uso de vectores se permite en todas las expresiones, sean aritméticas, de

asignación o relacionales, siendo su índice no limitado a variables o valores constantes sino por el contrario aceptando expresiones como vector[i*2+1]

El verificador semántico debe realizar las siguientes comprobaciones:

- No permitir el uso de variables no declaradas en programas o su uso de forma incorrecta, por ejemplo usar una variable simple como vector o viceversa.
- Permitir solo expresiones lógicas en las sentencia si(if), definiendo como expresión lógica aquella que puede devolver el valor verdadero o falso.
- Permitir solo ciclos para con el orden expresión de asignación, asignación, lógica en su interior.
- En cuanto a las variables (enteras o booleanas) se debe verificar la compatibilidad de tipos a la hora de llevar a cabo cualquier operación aritmética o asignación.
- Si se hace el llamado a una función esta debe existir (mirrarArriba por ejemplo).
- Este código debe ser llevado a cabo de forma manual en java como un recorrido especial extra al AST luego de su construcción y no mediante añadidos al código de CUP generado (analizador sintáctico).

NOTA:

- El trabajo debe ser presentado en equipos de máximo 4 personas, donde **TODOS** los integrantes **deben trabajar** en el proyecto y conocer como se hace cada parte del mismo, y debido a que la universidad además de enseñarles conocimientos técnicos, les debe enseñar algún tipo de valores personales a sus estudiantes, como por ejemplo una ética mínima y aún con más énfasis a los estudiantes que ya están tan cerca de culminar sus estudios y convertirse en futuros Ingenieros, el hecho de entregar un compilador cuyo funcionamiento interno EN **CUALQUIER ASPECTO** sea una incógnita para **UNO** o todos los integrantes de un equipo conllevara a la perdida de nota correspondiente al equipo (mínimo 50% de la nota del proyecto **A TODOS SUS INTEGRANTES**) y en caso de ser una falta considerada grave, se elevara el caso hasta las instancias correspondientes en la universidad para la aplicación del reglamento interno.