Parcial 1

Algoritmos y Estructuras de Datos

Pasolli, Nestor Jeremias 42853888

Natale, Alfredo 95558906

04/05/2022

AyED

PROBLEMA

Un archivo de texto contiene una serie de instrucciones. El pedido de la consigna es desarrollar un programa que las ejecute. Las instrucciones son de 5 tipos:

- declaración, (ej. “INT a”) crea una variable en la lista de variables con el tipo y nombre especificados , cada variable es identificada por un solo caracter alfabetico;
- asignación, (ej. “a = 2”) guarda el valor especificado en una variable declarada anteriormente;
- condicional, (ej. “IF (c < 2) THEN c = c + 3”) verifica la condición entre `IF` y `THEN` y si es verdadera ejecuta la instrucción que figura después de `THEN`;
- salto, (ej. “JUMP 7”) continúa la ejecución del programa a la línea especificada.

La consigna requiere que tanto instrucciones como variables sean almacenadas en listas.

Además las expresiones algebraicas del tipo “(3 + 2 * a) * 5” tienen que ser convertidas a notación postfija y posteriormente evaluadas.

DESARROLLO

En primer lugar comenzamos por plantear un diagrama de clases para nuestro programa, para orientarnos mejor con la consigna y tener un modelo base del funcionamiento del programa, para luego escribir el código más fácilmente.

El diagrama de clases planteado fue el que se muestra en la **imagen 1**. Como se observa, tenemos las clases correspondientes a Nodo, Pila y Lista, cuya implementación es la que se vio en clases, eliminando los métodos que no eran necesarios. Hay una clase procesador, que se encargará, como lo dice su nombre, del procesamiento de todas las instrucciones en el archivo de texto. Este archivo de texto será leído por la clase Lector, que luego enviará la lista de instrucciones al Procesador. La clase Instrucción representa una instrucción genérica, y de esta heredan las clases que representan cada tipo de instrucción en específico (Declaración, Asignación, Condicional, Jump, Show). Además, se tiene una clase llamada Variable, que se utilizará para representar las variables en juego en el archivo txt, y que serán almacenadas en una lista para luego leerlas o modificarlas. La clase Calculadora, por su parte, se encargará de los cálculos necesarios para procesar expresiones aritméticas y booleanas, incluyendo el pasaje de este tipo de expresiones de notación infija a posfija, operando finalmente esta expresión posfija en cada caso.

Terminado el proceso de modelado, procedimos a implementar estas clases y sus respectivos métodos, que dan funcionamiento a nuestro programa.

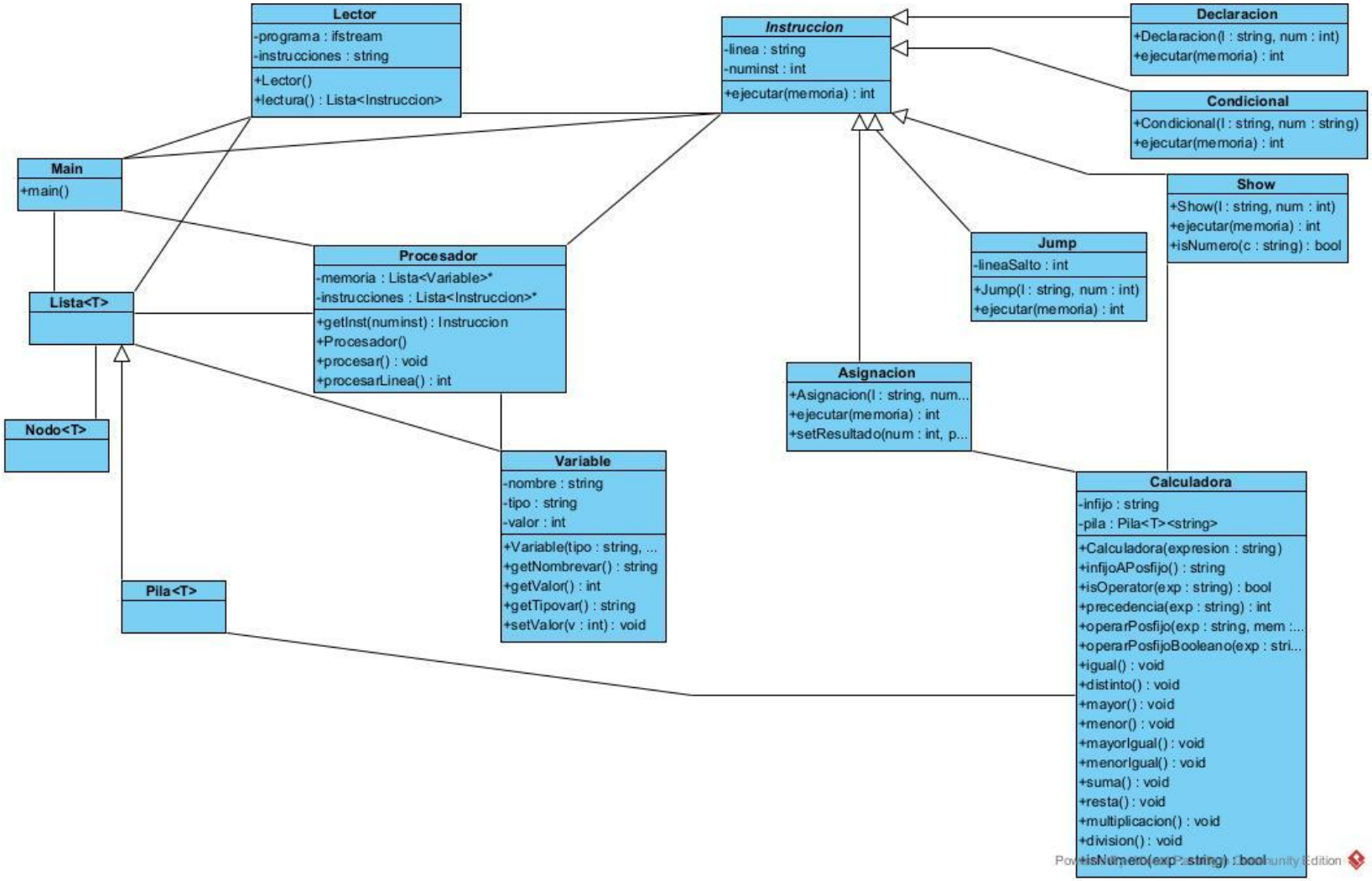
Lector
-programa : ifstream
-instrucciones : string
+Lector()
+lectura() : Lista<Instruccion>
Instruccion
-linea : string
-numinst : int
+ejecutar(memoria) : int
Declaracion
+Declaracion(l : string, num : int)
+ejecutar(memoria) : int
Condicional
+Condicional(l : string, num : string)
+ejecutar(memoria) : int
Main
+main()
Show
+Show(l : string, num : int)
+ejecutar(memoria) : int
+isNumero(c : string) : bool
Procesador
-memoria : Lista<Variable>*
-instrucciones : Lista<Instruccion>*
+getInst(numinst) : Instruccion
+Procesador()
+procesar() : void
+procesarLinea() : int
Jump
-lineaSalto : int
+Jump(l : string, num : int)
+ejecutar(memoria) : int
Lista<T>
Asignacion
+Asignacion(l : string, num...
+ejecutar(memoria) : int
+setResultado(num : int, p...
Nodo<T>
Variable
-nombre : string
-tipo : string
-valor : int
+Variable(tipo : string, ...
+getNombrevar() : string
+getValor() : int
+getTipovar() : string
+setValor(v : int) : void
Calculadora
-infijo : string
-pila : Pila<T><string>
+Calculadora(expresion : string)
+infijoAPosfijo() : string
+isOperator(exp : string) : bool
+precedencia(exp : string) : int
+operarPosfijo(exp : string, mem :...
+operarPosfijoBooleano(exp : stri...
+igual() : void
+distinto() : void
+mayor() : void
+menor() : void
+mayorIgual() : void
+menorIgual() : void
+suma() : void
+resta() : void
+multiplicacion() : void
+division() : void
+isNumero(exp : string) : bool
Pila<T>

Imagen 1

Método Main

El método main utiliza el Lector para leer el archivo de texto y obtener las instrucciones, que sucesivamente son pasadas al Procesador para que las ejecute.

```
int main(){
    Lector* lector = new Lector (); // crea un lector
    Lista<Instruccion>* listainst = lector->lectura(); // crea lista de instrucciones con retorno del lector
    Procesador* procesador = new Procesador (listainst); // crea procesador y le pasa la lista de instrucciones
    procesador->procesar(); // comienza a procesar el texto
    return 0;
}
```

Imagen 2

Clase Lector

Es la clase encargada de leer línea por línea el archivo de texto que contiene el programa, e ir almacenando todas las instrucciones.

```
class Lector{
public:
    Lista<Instruccion>* lectura(); // lee el txt, luego crea  y retorna una lista de instrucciones
private:
    ifstream programa; // se utiliza para abrir y leer el txt
    string instrucciones; // las lineas que lee programa se copian a este string
};
```

Imagen 3

Clase Procesador

Contiene la lista de variables y la lista de instrucciones, el método getInst(int) busca la instrucción correspondiente al número de línea en la lista. El método principal de Procesador es procesar(), que es el que empieza y dirige la ejecución del programa.

```
class Procesador {
public:
    Procesador (Lista<Instruccion>*); //constructor, recibe lista de instrucciones que genero el lector
    Instruccion getInst (int); // busca en la lista y retorna la instruccion correspondiente al
                               // numero de instruccion que se pasa como parametro
    void procesar(); // metodo principal, se encarga de procesar todas las instrucciones del texto
    int procesarLinea(Instruccion); // procesa solo una instruccion especifica

private:
    Lista<Instruccion>* instrucciones; // lista de instrucciones
    Lista<Variable*>* memoria; // lista donde se guardaran y asignaran variables luego
};
```

Imagen 4

Clase Variable

Esta clase representa una variable declarada en el programa, tiene tres miembros que indican su tipo, nombre y valor, con sus funciones getter asociadas, y una sola función setter para el valor, ya que una vez declaradas las variables no pueden cambiar su tipo y nombre.

```
class Variable {
private:
    string tipovar; // puede ser "int" o "bool"
    string nombrevar; // es una letra de la a "a" la "z"
    int valor; // un int cualquiera
public:
    Variable (string tipo="",string nombre="", int val = 0){tipovar = tipo; nombrevar = nombre; valor = val;};
                                            //Constructor, inicializa los atributos
    string getNombrevar(){return nombrevar;}; //getters y setters
    int getValor(){ return valor;};
    string getTipovar(){return tipovar;};
    void setValor(int v){ valor = v;};
};
```

Imagen 5

Clase Instrucción

Esta clase se encarga de representar de forma genérica cada instrucción del archivo de texto leído por el lector. Posee dos atributos, uno llamado “numinst”, que indica el número de la instrucción en cuestión (que se utilizará luego para ubicarla en la

lista de instrucciones), y otro llamado “linea”, que es la linea de texto leída en ese número de renglón del txt. En su método “ejecutar” se encarga de instanciar subclases según cómo se identifique la instrucción. Para esto utiliza varios métodos, que se detallan en la **imagen 6**.

```
class Instruccion {
public:
    Instruccion(string ="" , int=0); //construye instruccion con la linea de texto y el numero de
                                     // instruccion correspondiente
    virtual int ejecutar(Lista<Variable*>*); // deriva a cada tipo de instruccion segun sea el caso,
                                             // y retorna la siguiente instruccion a ejecutar
    int getNuminst(){return numinst;}; //retorna el numero de instruccion
    bool isJump(); //verifica si la linea es un JUMP
    bool isDeclaracion(); //verifica si la linea es una Declaracion
    bool isShow(); //verifica si la linea es un SHOW
    bool isCondicional(); //verifica si la linea es un Condicional
    bool isAsignacion(); //verifica si la linea es una Asignacion
    bool isBooleana(string); //Verifica si un string dado corresponde a una expresion booleana
protected:
    int numinst;
    string linea;
};
```

Imagen 6

Clase Declaración

Esta clase se instancia cuando una instrucción se identifica como Declaración, y se encarga de crear una nueva variable, con su tipo y nombre (le asigna “0” como valor), y de agregarla a la memoria (Lista de Variables). En la **imagen 7** se muestran sus métodos.

```
class Declaracion : public Instruccion{
public:
    Declaracion(string l, int num) : Instruccion(l, num) {}; //Constructor, llama a clase
                                                             // padre para incializar atributos
    int ejecutar(Lista<Variable*>*); // ejecuta declaracion
};
```

Imagen 7

Clase Show

Esta clase se instancia al interpretar una instrucción como SHOW, y simplemente imprime en pantalla lo que se le indique, ya sea un número, o una variable, o el resultado de una expresión aritmética o booleana. En la **imagen 8** se muestran sus métodos.

```
class Show : public Instruccion {
public:
    Show(string l, int num) : Instruccion(l, num) {};//Constructor, llama a clase
                                                     // padre para incializar atributos
    int ejecutar(Lista<Variable*>*); // hara una impresion de lo que haya luego de
                                     // la palabra SHOW en la linea
    bool isNumero(string); //identifica si se desea mostrar simplemente un numero
};
```

Imagen 8

Clase Asignación

Esta clase se instancia al interpretar una instrucción como Asignación, y se encarga de asignar valores a una variable dada, ya sean números, otras variables, o expresiones, que pueden ser algebraicas o booleanas. Para realizar las operaciones hace uso de la clase Calculadora. Los métodos de esta clase se encuentran en la **imagen 9**.

```
class Asignacion : public Instruccion{
public:
    Asignacion(string l, int num) : Instruccion(l, num) {};//Constructor, llama a clase
                                                           // padre para incializar atributos
    int ejecutar(Lista<Variable*>*); //asigna un numero, o una expresion, ya sea aritmetica
                                     //o booleana a una variable en la memoria
    void setResultado(int,string,Lista<Variable*>*);//se encargara se setear el resultado en
                                                    //la variable correcta de la memoria
};
```

Imagen 9

Clase Condicional

Esta clase se instancia al interpretar una instrucción como Condicional, y se encarga de analizar una expresión booleana luego de la palabra “IF”. Si esta resulta verdadera, ejecutará la instrucción que se encuentre luego de la palabra “THEN”, en caso

contrario simplemente no hará nada, continuando el procesador con la siguiente instrucción. Sus métodos se visualizan en la **imagen 10**.

```
class Condicional : public Instruccion {
public:
    Condicional(string l, int num) : Instruccion(l, num) {};//Constructor, llama a clase
                                                            // padre para incializar atributos
    int ejecutar(Lista<Variable*>*);// analiza la expresion booleana luego del "IF". Si
                                    // es verdadera, ejecuta la instruccion luego del
                                    //"THEN", caso contrario no hace nada
};
```

Imagen 10

Clase Calculadora

Esta clase resulta de suma importancia en el programa, ya que se encarga del pasaje de expresiones infijas a posfihas, como así también de operar luego las expresiones posfijas, sean booleanas o aritméticas. Es utilizada por las clases Asignación, Condicional, y Show eventualmente. Sus métodos se detallan en la **imagen 11**.

```
class Calculadora {
public:
    Calculadora(string); //Constructor, recibe exprecion infija como parametro
    string infijoAPosfijo ();//recibe expresion infija y la transforma a posfija
    bool isOperator(string);//verifica si un string es un operador
    int precedencia (string);//analiza precedencia de un operador dado
    int operarPosfijo(string,Lista<Variable*>*);//resuelve operacion aritmetica en posfijo
    int operarPosfijoBooleano(string,Lista<Variable*>*);//resuelve operacion booleana en posfijo
    void igual();
    void distinto();
    void mayor();
    void menor();
    void mayorIgual();
    void menorIgual();//Todos estos metodos evaluan expresiones booleanas, segun el operador,
                      // utilizando una pila
    void suma();
    void resta();
    void multiplicacion();
    void division(); // Estos hacen lo mismo, pero con expresiones arimeticas, segun el operador
    bool isNumero (string); //Analiza si un string corresponde a un numero
private:
    string infijo; //operacion en infijo
    Pila<string>* pila; // pila de strings que se utilzara para el pasaje de infijo a posfijo y
                        // para resolver las operaciones posfijas luego
};
```

Imagen 11

CONCLUSIONES

La realización de este proyecto resultó trabajosa, pero justamente por eso nos permitió afianzar los conocimientos sobre Listas y pilas, como así también el uso de punteros y la recursión.

Se utilizaron dos listas diferentes, una con objetos de la clase Instrucción, y otra con objetos de la clase Variable. Cabe aclarar que para la lista de variables se usaron punteros a objetos de tipo Variable, a diferencia de la lista de instrucciones, donde se usaron objetos estáticos. Esto es así porque en determinadas instrucciones necesitamos setear un valor específico a una variable, y si no se utilizan punteros a objetos Variable esto no se realizaba de forma correcta, ya que luego de salir del método setter respectivo

ese valor se perdía. Por eso se decidió usar punteros en este caso, resolviendo este problema.

Además investigamos y aprendimos sobre modalidades para dividir y procesar cadenas en C++, como así también a leer archivos de texto en formato .txt.

Restricciones del programa a tener en cuenta

El programa está diseñado para separar los strings de las instrucciones utilizando los espacios en blanco en estas, de modo que para su correcto funcionamiento se requiere que haya espacios entre una palabra y otra en el archivo de texto, como en el ejemplo del enunciado, de lo contrario el resultado no será el esperado. Lo mismo sucede si existen espacios en blanco luego de cada instrucción, o si hay líneas en blanco al final del archivo de texto. Si se evitan estas cuestiones, la interpretación del txt será correcta, según se ha testeado.