Universidad de San Carlos de Guatemala

Facultad de Ingeniería

Escuela de Ciencias y Sistemas

Organización de Lenguajes y Compiladores 2

Guillermo Alfredo Peitzner Estrada – 201504468.

Manual Técnico

Lenguaje utilizado

Python 3.8: https://www.python.org/.

Librerías

PyQt5 5.15.0: https://pypi.org/project/PyQt5/.

ply 3.11: https://pypi.org/project/ply/.

graphviz 0.14: https://pypi.org/project/graphviz/.

Dependencias de terceros

Graphviz: https://graphviz.org/.

Licencia del software

GPL-3.0: http://www.gnu.org/licenses/gpl-3.0.html

Repositorio oficial

https://github.com/gpeitzner/OLC2_Proyecto1

Archivos de código fuente utilizados

Nombre	Explicación
AMBITO.PY	Contiene la clase “Ámbito”, utilizada para los ámbitos dentro del lenguaje.
ASCENDENTE.PY	Contiene el analizador ascendente.
DESCENDENTE.PY	Contiene el analizador descendente.
EXPRESIONES.PY	Contiene la clase abstracta “Expresión” y clases que extienden esta, en las que se encuentran: “OperacionAritmetica”, “OperacioLogica”, “OperacionBit”, “OperacionRelacional”, “OperacionUnaria”, “OperacionFuncion” y “OperacionCasteo”. Este conjunto de clases es necesario para poder construir el AST en ambos analizadores.
INSTRUCCIONES.PY	Contiene la clase abstracta “Instrucción” y clases que extienden a esta, en las que se encuentran: “Principal”, “Etiqueta”, “Salto”, “Asignacion”, “AsignacionArreglo”, “Eliminar”, “Imprimir”, “Si” y “Salir”. Este conjunto de clases es necesario para poder construir el AST en ambos analizadores.
INTERPRETE.PY	Contiene al interprete que realiza la ejecución del AST generado por los analizadores.
PARSETAB.PY	Generado por PLY.
PRINCIPAL.PY	Es el archivo principal del proyecto, contiene la interfaz gráfica y su funcionalidad.
RECURSOS.PY	Archivo utilizado por la interfaz gráfica para la gestión de recursos como imágenes y contenido multimedia del proyecto.
TABLASIMBOLOS.PY	Contiene la definición de la tabla de símbolos del lenguaje.
TERROR.PY	Contiene a la clase “Terror”, que es la abstracción de un error léxico o sintáctico.
VALORES.PY	Contiene la abstracta “Valor” y clases que extienden a esta, en las que se encuentran: “Entero”, “Cadena”, “Carácter”, “Decimal”, “Registro”, “RegistroArreglo” y “Arreglo”. Este conjunto de clases es necesario para poder construir el AST en ambos analizadores.

Glosario de Términos

Python

Python es un lenguaje de programación interpretado cuya filosofía hace hincapié en la legibilidad de su código. Se trata de un lenguaje de programación multiparadigma, ya que soporta orientación a objetos, programación imperativa y, en menor medida, programación funcional.

PLY

PLY es una herramienta de análisis escrita exclusivamente en Python. Es, en esencia, una reimplementación de Lex y Yacc originalmente en lenguaje C. Fue escrito por David M. Beazley. PLY utiliza la misma técnica de análisis LALR que Lex y Yacc.

PyQT5

PyQt es un binding de la biblioteca gráfica Qt para el lenguaje de programación Python. La biblioteca está desarrollada por la firma británica Riverbank Computing y está disponible para Windows, GNU/Linux y Mac OS X bajo diferentes licencias.

Graphviz

Graphviz es un conjunto de herramientas de software para el diseño de diagramas definido en el lenguaje descriptivo DOT. Fue desarrollado por AT&T Labs y liberado como software libre con licencie tipo Eclipse.

Interprete

En ciencias de la computación, intérprete o interpretador es un programa informático capaz de analizar y ejecutar otros programas. Los intérpretes se diferencian de los compiladores o de los ensambladores en que mientras estos traducen un programa desde su descripción en un lenguaje de programación al código de máquina del sistema, los intérpretes sólo realizan la traducción a medida que sea necesaria, típicamente, instrucción por instrucción, y normalmente no guardan el resultado de dicha traducción.

Patrón interprete

Es un patrón de diseño que, dado un lenguaje, define una representación para su gramática junto con un intérprete del lenguaje. Se usa para definir un lenguaje para representar expresiones regulares que representen cadenas a buscar dentro de otras cadenas.

Fuente del glosario de términos: https://es.wikipedia.org/.

Gramática Ascendente

Producción	Reglas Semánticas
INIT -> main: CUERPO	INIT.val = Principal() + CUERPO.val
CUERPO -> INSTRUCCIONES	CUERPO.val = INSTRUCCIONES.val
CUERPO ->	CUERPO.val = None
INSTRUCCIONES -> INSTRUCCIONES INSTRUCCION	INSTRUCCIONES1.add(INSTRUCCION); INSTRUCCIONES.val = INSTRUCCIONES1
INSTRUCCIONES -> INSTRUCCION	INSTRUCCIONES.val = new list(INSTRUCCION)
INSTRUCCION -> ETIQUETA	INSTRUCCION.val = ETIQUETA.val
INSTRUCCION -> SALTO	INSTRUCCION.val = SALTO.val
INSTRUCCION -> ASIGNACION	INSTRUCCION.val = ASIGNACION.val
INSTRUCCION -> ASIGNACION_ARREGLO	INSTRUCCION.val = ASIGNACION_ARREGLO.val
INSTRUCCION -> ELIMINAR	INSTRUCCION.val = Eliminar.val
INSTRUCCION -> IMPRIMIR	INSTRUCCION.val = Imprimir.val
INSTRUCCION -> SI	INSTRUCCION.val = SI.val
INSTRUCCION -> SALIR	INSTRUCCION.val = SALIR.val
ETIQUETA -> identificador:	ETIQUETA.val = Etiqueta(identificador, t.lineno)
SALTO -> goto identificador;	SALTO.val = Salto(identificador, t.lineno)
ASIGNACION -> registro = EXPRESION;	ASIGNACION.val = Asignacion(registro, EXPRESION.val, t.lineno)
ASIGNACION_ARREGLO -> registro ACCESOS = EXPRESION;	ASIGNACION_ARREGLO.val = AsignacionArreglo(registro ', ACCESOS.val, EXPRESION.val, t.lineno)
ELIMINAR -> unset(EXPRESION);	ELIMINAR.val = Eliminar(EXPRESION.val, t.lineno)
IMPRIMIR -> print(EXPRESION);	IMPRIMIR.val = Imprimir(EXPRESION.val, t.lineno)
SI -> if (EXPRESION) goto identificador;	SI.val = Si(EXPRESION.val, identificador , t.lineno)
SALIR -> exit;	SALIR.val = Salir()
EXPRESION -> VALOR + VALOR **EXPRESION -> VALOR - VALOR** **EXPRESION -> VALOR * VALOR** **EXPRESION -> VALOR / VALOR** **EXPRESION -> VALOR % VALOR**	EXPRESION.val = OperacionAritmetica(VALOR.val, signo, VALOR.val)
EXPRESION -> VALOR && VALOR **EXPRESION -> VALOR || VALOR** **EXPRESION -> VALOR xor VALOR**	EXPRESION.val = OperacionLogica(VALOR.val, signo , VALOR.val)
EXPRESION -> VALOR & VALOR **EXPRESION -> VALOR | VALOR** **EXPRESION -> VALOR ^ VALOR** **EXPRESION -> VALOR << VALOR** **EXPRESION -> VALOR >> VALOR**	EXPRESION.val = OperacionBit(VALOR.val, signo, VALOR.val)

EXPRESION -> VALOR == VALOR **EXPRESION -> VALOR != VALOR** **EXPRESION -> VALOR >= VALOR** **EXPRESION -> VALOR <= VALOR** **EXPRESION -> VALOR > VALOR** **EXPRESION -> VALOR < VALOR**	EXPRESION.val = OperacionRelacional(VALOR.val, signo, VALOR.val)
EXPRESION -> - VALOR **EXPRESION -> & VALOR** **EXPRESION -> ! VALOR** **EXPRESION -> ~ VALOR**	EXPRESION.val = OperacionCasteo(signo, VALOR.val)
EXPRESION -> read() **EXPRESION -> abs(VALOR)** **EXPRESION -> array()**	EXPRESION.val = OperacionFuncion(función, VALOR.val)
EXPRESION -> (int) VALOR **EXPRESION -> (float) VALOR** **EXPRESION -> (char) VALOR**	EXPRESION.val = OperacionCasteo(casteo, VALOR.val)
EXPRESION -> VALOR	EXPRESION.val = VALOR.val
VALOR -> entero	VALOR.val = Entero(entero)
VALOR -> cadena	VALOR.val = Cadena(cadena)
VALOR -> caracter	VALOR.val = Caracter(caracter)
VALOR -> decimal	VALOR.val = Decimal(decimal)
VALOR -> registro	VALOR.val = Registro(registro)
ACCESOS -> ACCESOS ACCESO	ACCESOS.APPEND(ACCESO); ACCESOS.val = ACCESOS.val
ACCESOS -> ACCESO	ACCESOS.val = ACCESO.val
ACCESO -> [EXPRESION]	ACCESO.val = EXPRESION.val

Gramática Descendente

Producción	Reglas Semánticas
INIT -> main: CUERPO	INIT.val = Principal() + CUERPO.val
CUERPO -> INSTRUCCIONES	CUERPO.val = INSTRUCCIONES.val
CUERPO ->	CUERPO.val = None
INSTRUCCIONES -> INSTRUCCION INSTRUCCION_P	INSTRUCCIONES_P.add(INSTRUCCION); INSTRUCCIONES_P.reverse(); INSTRUCCIONES.val = INSTRUCCIONES_P
INSTRUCCIONES_P -> INSTRUCCION INSTRUCCIONES_P	INSTRUCCIONES_P1.add(INSTRUCCION); INSTRUCCIONES_P.val = INSTRUCCIONES_P1
INSTRUCCIONES_P ->	INSTRUCCIONES_P.val = new list()
INSTRUCCION -> ETIQUETA	INSTRUCCION.val = ETIQUETA.val
INSTRUCCION -> SALTO	INSTRUCCION.val = SALTO.val
INSTRUCCION -> ASIGNACION	INSTRUCCION.val = ASIGNACION.val
INSTRUCCION -> ASIGNACION_ARREGLO	INSTRUCCION.val = ASIGNACION_ARREGLO.val
INSTRUCCION -> ELIMINAR	INSTRUCCION.val = Eliminar.val
INSTRUCCION -> IMPRIMIR	INSTRUCCION.val = Imprimir.val
INSTRUCCION -> SI	INSTRUCCION.val = SI.val
INSTRUCCION -> SALIR	INSTRUCCION.val = SALIR.val
ETIQUETA -> identificador:	ETIQUETA.val = Etiqueta(identificador, t.lineno)
SALTO -> goto identificador;	SALTO.val = Salto(identificador, t.lineno)
ASIGNACION -> registro = EXPRESION;	ASIGNACION.val = Asignacion(registro, EXPRESION.val, t.lineno)
ASIGNACION_ARREGLO -> registro ACCESOS = EXPRESION;	ASIGNACION_ARREGLO.val = AsignacionArreglo(registro ', ACCESOS.val, EXPRESION.val, t.lineno)
ELIMINAR -> unset(EXPRESION);	ELIMINAR.val = Eliminar(EXPRESION.val, t.lineno)
IMPRIMIR -> print(EXPRESION);	IMPRIMIR.val = Imprimir(EXPRESION.val, t.lineno)
SI -> if (EXPRESION) goto identificador;	SI.val = Si(EXPRESION.val, identificador , t.lineno)
SALIR -> exit;	SALIR.val = Salir()
EXPRESION -> VALOR + VALOR **EXPRESION -> VALOR - VALOR** **EXPRESION -> VALOR * VALOR** **EXPRESION -> VALOR / VALOR** **EXPRESION -> VALOR % VALOR**	EXPRESION.val = OperacionAritmetica(VALOR.val, signo, VALOR.val)
EXPRESION -> VALOR && VALOR **EXPRESION -> VALOR || VALOR** **EXPRESION -> VALOR xor VALOR**	EXPRESION.val = OperacionLogica(VALOR.val, signo , VALOR.val)
EXPRESION -> VALOR & VALOR **EXPRESION -> VALOR | VALOR**	EXPRESION.val = OperacionBit(VALOR.val, signo, VALOR.val)

EXPRESION -> VALOR ^ VALOR **EXPRESION -> VALOR << VALOR** **EXPRESION -> VALOR >> VALOR**	
EXPRESION -> VALOR == VALOR **EXPRESION -> VALOR != VALOR** **EXPRESION -> VALOR >= VALOR** **EXPRESION -> VALOR <= VALOR** **EXPRESION -> VALOR > VALOR** **EXPRESION -> VALOR < VALOR**	EXPRESION.val = OperacionRelacional(VALOR.val, signo, VALOR.val)
EXPRESION -> - VALOR **EXPRESION -> & VALOR** **EXPRESION -> ! VALOR** **EXPRESION -> ~ VALOR**	EXPRESION.val = OperacionCasteo(signo, VALOR.val)
EXPRESION -> read() **EXPRESION -> abs(VALOR)** **EXPRESION -> array()**	EXPRESION.val = OperacionFuncion(función, VALOR.val)
EXPRESION -> (int) VALOR **EXPRESION -> (float) VALOR** **EXPRESION -> (char) VALOR**	EXPRESION.val = OperacionCasteo(casteo, VALOR.val)
EXPRESION -> VALOR	EXPRESION.val = VALOR.val
VALOR -> entero	VALOR.val = Entero(entero)
VALOR -> cadena	VALOR.val = Cadena(cadena)
VALOR -> caracter	VALOR.val = Caracter(caracter)
VALOR -> decimal	VALOR.val = Decimal(decimal)
VALOR -> registro	VALOR.val = Registro(registro)
ACCESOS -> ACCESO INSTRUCCION_P	ACCESOS_P.add(ACCESO); ACCESOS_P.reverse(); ACCESOS.val = ACCESOS_P
ACCESOS_P -> ACCESO ACCESOS_P	ACCESOS_P1.add(ACCESO); ACCESOS_P.val = ACCESOS_P1
ACCESOS_P ->	ACCESOS_P.val = new list()
ACCESO -> [EXPRESION]	ACCESO.val = EXPRESION.val