Pontificia Universidad Católica de Chile
Escuela de Ingeniería
Departamento de Ciencia de la Computación

IIC3745 — Testing
2023 - 1

Tarea 1

Fecha de entrega: Martes 28 de Marzo del 2023 a las 23:59

Información General

La siguiente tarea contempla como objetivo principal la familiarización con los conceptos de parsing, árbol de sintaxis abstracta, el patrón composite, y el patrón visitor. Recordar que existe un foro en canvas donde el equipo de ayudantes estará atento para resolver dudas con respecto a la tarea.

Objetivos

- Comprender los conceptos de parsing y árbol de sintaxis abstracta.
- Familiarizarse con el patrón visitor y composite.
- Crear tu primer analizador de código estático.

Contexto

Considere el siguiente modelo de clases:

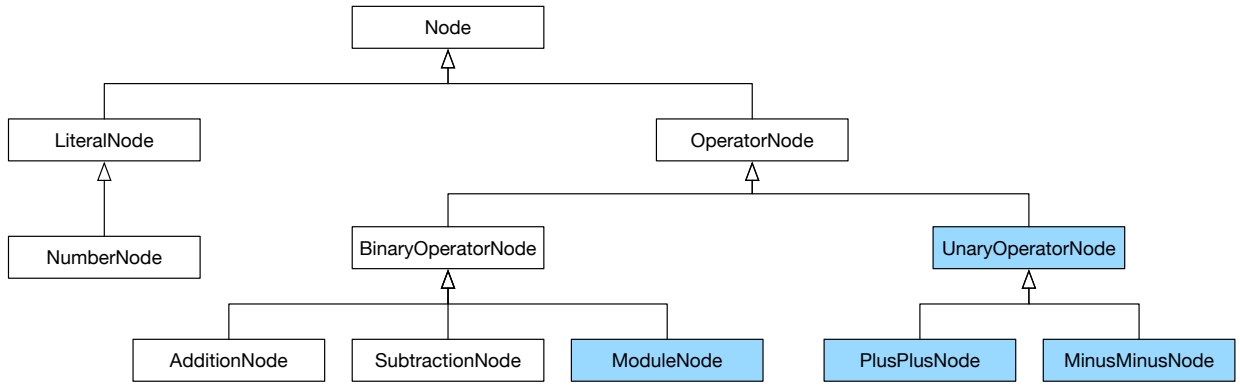

Figura 1: Diagrama Clases

Los nodos en color blanco son las clases implementadas en el curso de Testing. Estas clases permiten modelar en forma de un árbol las componentes de un programa de sintaxis sencilla que solo soportaba operaciones aritméticas de adición y sustracción. Note que la gramática actual está denotada en backus normal form (BNF) de la siguiente manera:

```
<expr> =  <number>
       |  (+ <expr> <expr>)
       |  (- <expr> <expr>)
```

Por lo tanto, varias expresiones pueden ser formadas, por ejemplo:

```
(+ 1 1)
(+ (- 2 1) (+ 2 2))
```

En esta tarea se quiere agregar tres operaciones al lenguaje: %, ++ y - -. Después de agregar estas operaciones, el lenguaje podría soportar las siguientes expresiones:

```
(++ 1)
(+ (-- 5) (% 2 8))
(+ 7 (++ 10))
(% 7 4)
entre otras posibles combinaciones
```

Y la gramática quedará de la siguiente forma:

```
<expr> =  <number>
       |  (+ <expr> <expr>)
       |  (- <expr> <expr>)
       |  (% <expr> <expr>)
       |  (++ <expr>)
       |  (-- <expr>)
```

Ejercicio 1

Agregue la operación binaria % al lenguaje:

- **(a)** Implemente la clase `ModuloNode` que hereda de la clase `BinaryOperatorNode`. También implemente los métodos `__init__` y `eval` en la clase `ModuloNode`.
- **(b)** Modifique las funciones del archivo `parser.py` creado en clase de tal forma que se pueda parsear expresiones con los operadores binarios creados en clase y el implementado en el inciso (a).
- **(c)** Modifique la clase `Visitor`, y agregue el método `visit_Modulo`. Adicionalmente, agregue el método `accept(self, visitor)` en la clase `ModuloNode`.
- **(d)** Agregue una nueva clase llamada `ModuloOperatorCounter`, la misma debe heredar de la clase `Visitor`. Esta clase debe visitar todos los nodos del árbol y contar el número de operadores % presentes en una expresión.

Ejercicio 2

Agregue las operaciones unarias ++ y - - al lenguaje.

- **(a)** Implemente las clases `UnaryOperatorNode`, `PlusPlusNode`, `MinusMinusNode`. También implemente los métodos `__init__`, `__eq__`, `eval` y `to_string` según corresponda en las clases mencionadas.
- **(b)** Modifique las funciones del archivo `parser.py` creado en clase de tal forma que se pueda parsear expresiones con los operadores unarios (los agregados en el inciso (a)) y binarios (los creados en clase y en el ejercicio 1).
- **(c)** Modifique la clase `Visitor`, y agregue los métodos `visit_PlusPlus` y `visit_MinusMinus`. Además, agregue el método `accept(self, visitor)` en las clases creadas en el inciso (a).
- **(d)** Agregue una nueva clase llamada `UnaryOperatorCounter`, la misma debe heredar de la clase `Visitor`. Esta clase debe visitar todos los nodos del árbol y contar el número de operadores unarios presentes en una expresión, tanto `++` y `--`.

Nota: Si bien la tarea viene con algunos test de referencia, usted puede agregar mas tests para asegurar que su implementación funcione como espera. Por ejemplo, probar diferentes tipos de expresiones. Tenga en cuenta que las expresiones deben ser validas según la gramática.

Revisión

Considere los siguientes test de ejemplo:

```
class TestParser(unittest.TestCase):
    # Tests para el ejercicio 1
    def test_mn(self):
        ast1 = ModuleNode(NumberNode(5), NumberNode(2))
        ast2 = parser("(% 5 2)")
        self.assertEqual(ast1,ast2)
    def test_mn_eval(self):
        ast = parser("(% 5 2)")
        result = ast.eval()
        self.assertEqual(result,1)
    def test_mix_bin(self):
        ast = parser("(% (+ 35 15) (- 30 8))")
        result = ast.eval()
        self.assertEqual(result,6)
    def test_to_string_mn(self):
        ast1 = ModuleNode(AdditionNode(NumberNode(2), NumberNode(8)), SubstractNode(NumberNode(7), NumberNode(4)))
        self.assertEqual(ast1.to_string(),"(% (+ 2 8) (- 7 4))")
    def test_to_string_mn2(self):
        ast1 = AdditionNode(ModuleNode(NumberNode(7),NumberNode(2)), ModuleNode(NumberNode(13), NumberNode(2)))
        self.assertEqual(ast1.to_string(),"(+ (% 7 2) (% 13 2))")
    def test_module_counter(self):
        visitor = ModuleOperatorCounter()
        ast = parser("(% (% 30 11) (% 65 8))")
        ast.accept(visitor)
        self.assertEqual(visitor.total(), 3)
    def test_module_counter2(self):
        visitor = ModuleOperatorCounter()
        ast = parser("(% (+ 6 1) (% 8 5))")
        ast.accept(visitor)
        self.assertEqual(visitor.total(), 2)
    # Tests para el ejercicio 2
    def test_pp(self):
        ast1 = PlusPlusNode(NumberNode(2))
        ast2 = parser("(++ 2)")
        self.assertEqual(ast1,ast2)
    def test_pp_eval(self):
        ast = parser("(++ 2)")
        result = ast.eval()
        self.assertEqual(result,3)
    def test_mm_eval(self):
        ast = parser("(-- 3)")
        result = ast.eval()
        self.assertEqual(result,2)
    def test_mix(self):
        ast = parser("(+ (++ 1) (++ 1))")
        result = ast.eval()
        self.assertEqual(result,4)
    def test_to_string2(self):
        ast1 = PlusPlusNode(MinusMinusNode(NumberNode(2)))
        self.assertEqual(ast1.to_string(),"(++ (-- 2))")
    def test_to_string3(self):
        ast1 = AdditionNode(PlusPlusNode(NumberNode(1)),MinusMinusNode(NumberNode(2)))
        self.assertEqual(ast1.to_string(),"(+ (++ 1) (-- 2))")
    def test_unary_counter(self):
        visitor = UnaryOperatorCounter()
        ast = parser("(+ (+ (++ 1) (++ 1)) (- 2 (-- 3)))")
        ast.accept(visitor)
        self.assertEqual(visitor.total(), 3)
    def test_unary_counter2(self):
        visitor = UnaryOperatorCounter()
        ast = parser("(++ 1)")
        ast.accept(visitor)
        self.assertEqual(visitor.total(), 1)
```

Los ayudantes tienen una batería de pruebas, donde se evalúan los métodos solicitados con diferentes tipos de expresiones. La nota se asignará con respecto al número de test que pasen.

- **(3 puntos)** Los tests entregados para cada ejercicio pasan y el código no está hardcodeado (diseñado solo para pasar los tests). Se asignará un puntaje entre 0 y 3, en base al número de tests que pasen.
- **(3 puntos)** Se ejecutarán un conjunto de tests adicionales creado por los ayudantes para el ejercicio 1 y 2. Se asignará un puntaje entre 0 y 3, en base al número de tests adicionales que pasen.

Archivos iniciales

En canvas se encuentra un conjunto de archivos que se implementaron en clases:

- *parser.py* – que contiene las funciones necesarias para parsear.
- *model.py* – que contiene las clases correspondientes al árbol de sintaxis abstracta.
- *metrics.py* – que contiene los visitors creados, donde cada uno computa una métrica.
- *test_clases.py* – que contiene el código de prueba desarrollado en clase.
- *test_ejemplo_tarea.py* – que contiene las pruebas de los ejercicios 1 y 2 del enunciado de la tarea.

Para la tarea usted debe modificar o agregar código a los archivos anteriormente mencionados según corresponda.

Reportar problemas en el equipo

En el caso de que algún integrante no aportara como fue esperado en la tarea, podrán reportarlo enviando un correo con asunto **Problema Equipo {NumeroGrupo} Testing** a *juanandresarriagada@uc.cl* con copia a *afernandb@uc.cl* explicando en detalle lo ocurrido. Posterior a eso se revisará el caso en detalle con los involucrados y se analizará si corresponde aplicar algún descuento. Instamos a todas las parejas que mantengan una buena comunicación y sean responsables con el resto de su equipo para evitar problemas de este estilo.

Advertencia: Si algun integrante del grupo no aporta en las tareas puede implicar recibir nota mínima en esa entrega.

Restricciones y alcances

- Su programa debe ser desarrollado en **Python 3.9 o superior**.
- Los archivos entregados deben terminar con la extensión **.py**.
- En caso de dudas con respecto al enunciado deben realizarlas en un foro relacionado a la tarea que se encontrará disponible en canvas.
- Si no se encuentra especificado en el enunciado, supón que el uso de cualquier librería de Python adicional a las utilizadas en el código base se encuentran prohibidas. En caso de que estimes necesario podrás preguntar en el foro de la tarea por el uso de alguna librería adicional.

Entrega

Código: Deberán entregar todos los archivos iniciales a excepción de los archivos de tests. La entrega se realizara por medio de un buzón de canvas habilitado para esta tarea.

Declaración de tarea: Deberán entregar un archivo README que contiene los nombres de quienes realizaron la tarea y su aporte. La entrega de este archivo se realizara junto con el código por medio de un buzón de canvas habilitado para esta tarea.

Atraso: Se efectuará un descuento por entregar tareas atrasadas. Se descontará 0.5 si la tarea se entrega con menos de una hora de retraso. El puntaje final en caso de atraso será calculado mediante la siguiente fórmula:

$$PuntajeFinal = PuntajeObtenido - (0,5 + 0,05 \cdot k)$$

, donde k es el número de horas de retraso menos uno.

Integridad académica

Este curso se adscribe al Código de Honor establecido por la Escuela de Ingeniería. Todo trabajo evaluado en este curso debe ser hecho **individualmente** o en **los grupos asignados** según sea definido en la evaluación y **sin apoyo de terceros**. Se espera que cada estudiante mantenga altos estándares de honestidad académica, acorde al Código de Honor de la Universidad. Cualquier acto deshonesto o fraude académico está prohibido; quien incurra en este tipo de acciones se expone a un Procedimiento Sumario. Es responsabilidad de cada estudiante conocer y respetar el documento sobre Integridad Académica publicado por la Dirección de Pregrado de la Escuela de Ingeniería.
¡Éxito! :)