Report



과 목 명 : 컴파일러

담당교수님 : 우균

학 번 : 201324474

이 름 : 안도현

1. **Instruction structure**
   1. **명령**

* 본 과제의 구현물은 아래 입력 구조를 따른다.

*{Instruction} {Number}*

* 모든 입력은 명령어를 입력한 후 명령 인자인 숫자를 연속해서 입력해야 한다.
* 제공되는 명령은 아래와 같다

F : 앞으로 전진한다. Turtle의 forward와 같다.

R : 오른쪽으로 회전한다. Turtle의 right와 같다

L : 왼쪽으로 회전한다. Turtle의 left와 같다.

* 1. **사용 예시**

F12R44F45L12F10

위 명령은 다음 파이썬 스크립트의 기능을 수행한다.

t = Turtle()

t.forward(12)

t.right(44)

t.forward(45)

t.left(12)

t.forward(10)

1. **Interpreter**
   1. **사용법**

python interpreter.py [input]

* 본 스크립트는 파이썬 Turtle 패키지를 좀 더 간결한 명령을 통해 실행할 수 있도록 하는 언어를 구현하고, 해당 언어의 interpreter를 제공한다.
* Input 파일을 별도로 제공함으로 미리 입력한 명령을 실행할 수 있다. 미리 입력한 명령을 모두 수행하였다면 명령줄을 통해 추가 입력을 받을 수 있다.
  1. **구현**
     1. **Main**

1. # read file and interpretation if given
2. if len(sys.argv) > 1:
3. file = sys.argv[1]
4. fd = open(str(sys.argv[1]), 'r')
6. while True:
7. inp = fd.readline()
8. if not inp: break
9. parser(inp)
11. fd.close()
13. # interpreter mode
14. while True:
15. print("> ", end="")
16. inp = input()
17. if inp == "exit": break
18. parser(inp)

* 사용자로부터 입력을 받고, 해당 내용을 parser로부터 전달해주는 역할을 한다.
* 스크립트 실행 시 input이 주어졌다면 해당 스크립트를 먼저 실행한다. 이후 명령줄을 통해 추가 입력을 받을 수 있다.
* 사용자로부터 “exit”을 입력 받을 때까지 사용자가 입력한 명령을 수행한다.

* + 1. **Parser**

1. def parser(inp):
2. global q
3. q.queue.clear()
4. mode = Mode.inst
5. cnt = ''
6. inst = ''
8. str = list(inp)
9. leng = len(str)
10. for i in range(leng):
11. c = str[0]
12. str.pop(0)
14. if c.isalpha():
15. # case 1. Last character is inst or something else
16. if (len(str) == 0):
17. print('ErrorSyntax error near input ' + c + '. Can\'t find pair number.\n')
18. printError(inp, i)
19. return
21. # case 2. Continuous instructions
22. if mode == Mode.num:
23. print(
24. 'Syntax error near instruction "' + inst + '". Instructions can\'t be continuous. Given input was ' + c + "\n")
25. printError(inp, i)
26. return
28. # Next instruction given. push current instruction to q
29. elif mode == Mode.both:
30. q.put((inst, cnt))
31. cnt = ''
32. inst = ''
33. mode = Mode.inst
35. if c == "F":
36. inst = "F"
37. elif c == "L":
38. inst = "L"
39. elif c == "R":
40. inst = "R"
42. # case 3. can't match any instructions.
43. else:
44. printError(inp, i)
45. print('Syntax error near ' + c + '. Not supported instruction.\n')
46. return
47. mode = Mode.num
49. elif c.isdigit():
50. # case 4. First input was number.
51. if mode == Mode.inst:
52. printError(inp, i)
53. print('Syntax error near ' + c + '. No instruction given.\n')
54. return
56. cnt += c
58. if (len(str) == 0):
59. q.put((inst, cnt))
60. mode = Mode.both
61. else:
62. # case 5. Can't find such syntax.
63. printError(inp, i)
64. print('Syntax error near ' + c + '. Invalid input.\n')
65. return
67. goTurtle()

* 사용자의 입력이 문법에 맞는지 확인하고 실행 내역을 정리한다.
* 실행내역은 전역변수로 선언된 큐에 들어온 순서대로 삽입한다.
* 사용자의 입력을 한 문자씩 읽어와 명령과 명령 인자를 처리한다.
* 최초 메서드 진입 시 inst 모드로 설정된다. 해당 모드는 다음 읽어와야 하는 문자가 명령 인자라는 것을 의미한다.
* 명령을 읽고 나면 num 모드로 설정된다. 해당 모드는 다음 읽어와야 하는 문자가 명령이라는 것을 의미한다.
* 명령 인자를 읽고 오면 both 모드로 설정된다. 해당 모드는 다음 문자가 명령이나 명령 인자 둘 다 될 수 있다는 것을 의미한다.
* 하나의 명령과 명령 인자를 읽었다면 해당 내용을 pair 타입으로 작업 큐에 추가한다.
* 이후 goTurtle() 메서드를 호출하여 실제 함수를 실행한다.
* Parser는 다음과 같은 입력 예외를 처리한다.
  1. 마지막으로 입력된 문자가 숫자가 아닐 경우
* 명령은 항상 명령 인자와 함께 입력되어야 한다.
  1. 명령이 입력되고 인자가 아닌 명령이 입력된 경우
* 명령은 연속되게 입력될 수 없다. 항상 명령 인자와 함께 입력되어야 한다.
  1. 사용자가 입력한 문자가 명령이 아닌 경우
* 1.1에서 서술한 명령 이외는 지원하지 않는다.
  1. 명령이 입력되지 않고 인자만 입력된 경우
* 인자 단독으로 사용할 수 없다.
  1. 알파벳이나 숫자가 아닌 문자가 입력된 경우
* 명령과 명령 인자인 숫자 이외에는 사용할 수 없다.
  + 1. **Execute**

1. global q
2. while q.qsize() > 0:
3. inst, num = q.queue.pop()
4. num = int(num)
6. if inst == "F":
7. t.forward(num)
8. elif inst == "R":
9. t.right(num)
10. else:
11. t.left(num)

* Parser에서 큐에 삽입한 명령을 실행한다.
  + 1. **etc**

1. def printError(inp, count):
2. print("---------------------------------------------------------------------------")
3. print("Error position :")
4. print(" " + inp)
5. print(" " + " " \* count + "↑")
6. print("---------------------------------------------------------------------------\n")

* printError 메서드는 사용자의 입력에 문제가 있을 경우 입력 내용 중 어떤 부분이 문제를 발생시켰는지 알려준다.

1. **Compiler**
   1. **사용법**

python compiler.py [input]

* 본 스크립트는 파이썬 Turtle 패키지를 좀 더 간결한 명령을 통해 실행할 수 있도록 하는 언어를 구현하고, 해당 언어의 compiler를 제공한다.
* Input 파일을 별도로 제공함으로 미리 입력한 명령을 실행할 수 있다.
* 컴파일 결과물은 같은 디렉토리의 a.out.py 파일로 저장된다.
* 컴파일 된 내용의 포맷은 다음과 같다.

1. from turtle import \*
2. def main():
3. t = Turtle()
4. ... # Translated user input
5. exitonclick()
6. if \_\_name\_\_ == '\_\_main\_\_':
7. main()

* 1. **구현**
     1. **Main**

1. result = ''
3. # read from file mode
4. if len(sys.argv) > 1:
5. file = sys.argv[1]
6. fd = open(str(sys.argv[1]), 'r')
8. while True:
9. inp = fd.readline()
10. if not inp: break
12. tmp = parser(inp)
13. if isinstance(tmp, NoneType):
14. continue
15. result += parser(inp)
17. fd.close()
19. c = compiler()
20. c.compile(result)
22. # read from stdin mode
23. else:
24. while True:
25. print("> ", end="")
26. inp = input()
27. if inp == "exit" :
28. c = compiler()
29. c.compile(result)
30. break
32. tmp = parser(inp)
33. if isinstance(tmp, NoneType):
34. continue
35. else:
36. result += tmp
37. q.queue.clear()

* 사용자로부터 입력을 받고, 해당 내용을 parser로부터 전달해주는 역할을 한다.
* 스크립트 실행 시 input이 주어졌다면 해당 스크립트를 컴파일 한다. Interpreter와 달리 이후 추가 명령을 입력할 수 없다.
* 명령줄 모드로 실행될 경우 사용자로부터 “exit”을 입력 받을 때까지 사용자가 입력한 명령을 수행한다. 이후 입력된 내용을 컴파일한다.
* 사용자의 입력을 파싱을 거친 후 컴파일한다. 이 때 compile 클래스[[1]](#footnote-1)를 사용한다.
* 파싱 단계에서 오류나 예외가 발생할 경우, 컴파일을 수행하지 않는다.

* + 1. **Parser**
* 사용자로부터 입력을 받고, 해당 내용을 점검한 후 goTurtle 메서드로부터 메인까지 전달해주는 역할을 한다.
* 기본적인 구현은 Interpreter와 동일하며, goTurtle[[2]](#footnote-2) 메서드로부터 리턴받은 값을 그대로 리턴한다.
  + 1. **Translate**

1. def goTurtle():
2. global q
3. ir = ''
5. while q.qsize() > 0:
6. inst, num = q.queue.pop()
8. if inst == "F":
9. ir += ' t.forward(' + num + ')\n'
10. elif inst == "R":
11. ir += ' t.right(' + num + ')\n'
12. else:
13. ir += ' t.left(' + num + ')\n'
15. return ir

* parser에서 큐에 삽입한 내용을 바탕으로 사용자의 입력을 파이썬 스크립트로 번역하는 역할을 한다. 컴파일러의 IR과 비슷한 기능을 한다.
* 번역된 결과물을 parser로 리턴한다.
  + 1. **Compiler Class**

1. class compiler:
2. def \_\_init\_\_(self):
3. self.prologue = "from turtle import \*\n" \
4. "def main():\n" \
5. " t = Turtle()\n"
6. self.epilogue = " t.hideturtle()\n" \
7. " exitonclick()\n" \
8. "if \_\_name\_\_ == '\_\_main\_\_':\n" \
9. " main()\n"
10. self.middle = ''
12. def compile(self, ir):
13. self.middle += self.prologue
14. self.middle += ir
15. self.middle += self.epilogue
17. fd = open("a.out.py", "w")
18. fd.write(self.middle)
19. fd.close()

* 해당 클래스는 전달받은 번역물을 바탕으로 파이썬 스크립트를 생성한다.
* 에필로그와 프롤로그로 생성될 파이썬 스크립트를 구성한다.
* goTurtle 메서드로부터 생성된 중간 결과물을 프롤로그와 에필로그 사이에 삽입한다.

1. **Source Code**
   1. **Interpreter code**
2. from turtle import \*
3. import sys, enum, queue

6. class Mode(enum.Enum):
7. inst = 0
8. num = 1
9. both = 2

12. t = Turtle()
13. q = queue.Queue()

16. def goTurtle():
17. global q
18. while q.qsize() > 0:
19. inst, num = q.queue.pop()
20. num = int(num)
22. if inst == "F":
23. t.forward(num)
24. elif inst == "R":
25. t.right(num)
26. else:
27. t.left(num)

30. def printError(inp, count):
31. print("---------------------------------------------------------------------------")
32. print("Error position :")
33. print(" " + inp)
34. print(" " + " " \* count + "↑")
35. print("---------------------------------------------------------------------------\n")

38. def parser(inp):
39. global q
40. q.queue.clear()
41. mode = Mode.inst
42. cnt = ''
43. inst = ''
45. str = list(inp)
46. leng = len(str)
47. for i in range(leng):
48. c = str[0]
49. str.pop(0)
51. if c.isalpha():
52. # case 1. Last character is inst or something else
53. if (len(str) == 0):
54. print('ErrorSyntax error near input ' + c + '. Can\'t find pair number.\n')
55. printError(inp, i)
56. return
58. # case 2. Continuous instructions
59. if mode == Mode.num:
60. print(
61. 'Syntax error near instruction "' + inst + '". Instructions can\'t be continuous. Given input was ' + c + "\n")
62. printError(inp, i)
63. return
65. # Next instruction given. push current instruction to q
66. elif mode == Mode.both:
67. q.put((inst, cnt))
68. cnt = ''
69. inst = ''
70. mode = Mode.inst
72. if c == "F":
73. inst = "F"
74. elif c == "L":
75. inst = "L"
76. elif c == "R":
77. inst = "R"
79. # case 3. can't match any instructions.
80. else:
81. printError(inp, i)
82. print('Syntax error near ' + c + '. Not supported instruction.\n')
83. return
84. mode = Mode.num
86. elif c.isdigit():
87. # case 4. First input was number.
88. if mode == Mode.inst:
89. printError(inp, i)
90. print('Syntax error near ' + c + '. No instruction given.\n')
91. return
93. cnt += c
95. if (len(str) == 0):
96. q.put((inst, cnt))
97. mode = Mode.both
98. else:
99. # case 5. Can't find such syntax.
100. printError(inp, i)
101. print('Syntax error near ' + c + '. Invalid input.\n')
102. return
104. goTurtle()

107. def solver():
108. fd = sys.stdin
110. # read file and interpretation if given
111. if len(sys.argv) > 1:
112. file = sys.argv[1]
113. fd = open(str(sys.argv[1]), 'r')
115. while True:
116. inp = fd.readline()
117. if not inp: break
118. parser(inp)
120. fd.close()
122. # interpreter mode
123. while True:
124. print("> ", end="")
125. inp = input()
126. if inp == "exit": break
127. parser(inp)

130. if \_\_name\_\_ == '\_\_main\_\_':
131. if (sys.platform == "win32" or sys.platform == "win64"):
132. import os, msvcrt
134. msvcrt.setmode(sys.stdin.fileno(), os.O\_BINARY)
135. solver()

* 1. **Compiler code**

1. from turtle import \*
2. import sys, enum, queue

5. class Mode(enum.Enum):
6. inst = 0
7. num = 1
8. both = 2
10. class compiler:
11. def \_\_init\_\_(self):
12. self.prologue = "from turtle import \*\n" \
13. "def main():\n" \
14. " t = Turtle()\n"
15. self.epilogue = " t.hideturtle()\n" \
16. " exitonclick()\n" \
17. "if \_\_name\_\_ == '\_\_main\_\_':\n" \
18. " main()\n"
19. self.middle = ''
21. def compile(self, ir):
22. self.middle += self.prologue
23. self.middle += ir
24. self.middle += self.epilogue
26. fd = open("a.out.py", "w")
27. fd.write(self.middle)
28. fd.close()
30. q = queue.Queue()
31. output = ''
32. NoneType = type(None)
34. def goTurtle():
35. global q
36. ir = ''
38. while q.qsize() > 0:
39. inst, num = q.queue.pop()
41. if inst == "F":
42. ir += ' t.forward(' + num + ')\n'
43. elif inst == "R":
44. ir += ' t.right(' + num + ')\n'
45. else:
46. ir += ' t.left(' + num + ')\n'
48. return ir
50. def printError(inp, count):
51. print("---------------------------------------------------------------------------")
52. print("Error position :")
53. print(" " + inp)
54. print(" " + " " \* count + "↑")
55. print("---------------------------------------------------------------------------\n")

58. def parser(inp):
59. global q
60. q.queue.clear()
61. mode = Mode.inst
62. cnt = ''
63. inst = ''
65. str = list(inp)
66. leng = len(str)
67. for i in range(leng):
68. c = str[0]
69. str.pop(0)
71. if c.isalpha():
72. # case 1. Last character is inst or something else
73. if (len(str) == 0):
74. print('ErrorSyntax error near input ' + c + '. Can\'t find pair number.\n')
75. printError(inp, i)
76. return
78. # case 2. Continuous instructions
79. if mode == Mode.num:
80. print(
81. 'Syntax error near instruction "' + inst + '". Instructions can\'t be continuous. Given input was ' + c + "\n")
82. printError(inp, i)
83. return
85. # Next instruction given. push current instruction to q
86. elif mode == Mode.both:
87. q.put((inst, cnt))
88. cnt = ''
89. inst = ''
90. mode = Mode.inst
92. if c == "F":
93. inst = "F"
94. elif c == "L":
95. inst = "L"
96. elif c == "R":
97. inst = "R"
99. # case 3. can't match any instructions.
100. else:
101. printError(inp, i)
102. print('Syntax error near ' + c + '. Not supported instruction.\n')
103. return
104. mode = Mode.num
106. elif c.isdigit():
107. # case 4. First input was number.
108. if mode == Mode.inst:
109. printError(inp, i)
110. print('Syntax error near ' + c + '. No instruction given.\n')
111. return
113. cnt += c
115. if (len(str) == 0):
116. q.put((inst, cnt))
117. mode = Mode.both
118. else:
119. # case 5. Can't find such syntax.
120. printError(inp, i)
121. print('Syntax error near ' + c + '. Invalid input.\n')
122. return
124. return goTurtle()

127. def solver():
128. result = ''
130. # read from file mode
131. if len(sys.argv) > 1:
132. file = sys.argv[1]
133. fd = open(str(sys.argv[1]), 'r')
135. while True:
136. inp = fd.readline()
137. if not inp: break
139. tmp = parser(inp)
140. if isinstance(tmp, NoneType):
141. continue
142. result += parser(inp)
144. fd.close()
146. c = compiler()
147. c.compile(result)
149. # read from stdin mode
150. else:
151. while True:
152. print("> ", end="")
153. inp = input()
154. if inp == "exit" :
155. c = compiler()
156. c.compile(result)
157. break
159. tmp = parser(inp)
160. if isinstance(tmp, NoneType):
161. continue
162. else:
163. result += tmp
164. q.queue.clear()

167. if \_\_name\_\_ == '\_\_main\_\_':
168. if (sys.platform == "win32" or sys.platform == "win64"):
169. import os, msvcrt
170. msvcrt.setmode(sys.stdin.fileno(), os.O\_BINARY)
171. solver()

1. 3.2.4. Compile Class [↑](#footnote-ref-1)
2. 3.2.3. Translate [↑](#footnote-ref-2)