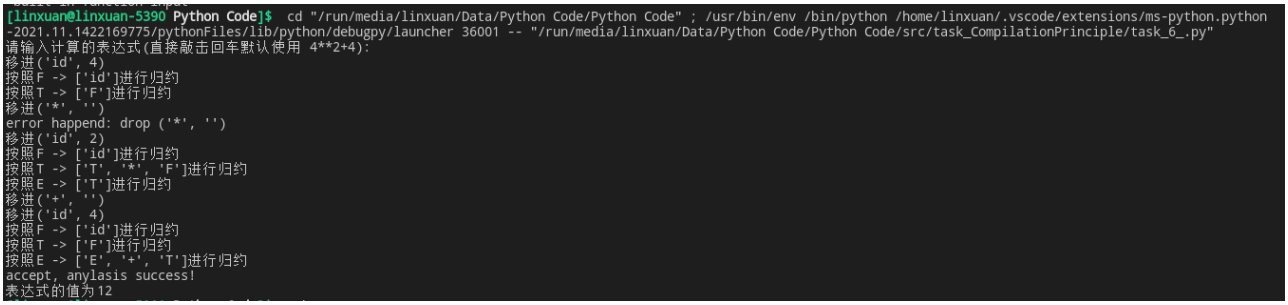
第七次编译上机作业

运行截图



运行说明：

直接运行文件，按照提示输入计算表达式即可

代码：

```python
'''
Author: LinXuan
Date: 2021-11-09 20:32:16
Description: 改造上一次的LR分析，使其支持翻译方案
LastEditors: LinXuan
LastEditTime: 2021-12-06 22:46:13
FilePath: /Python Code/src/task_CompilationPrinciple/task_6_.py
'''
import re
import json

# 构造分析表


def get_table():
    talbe_json = '''{
        "I0": { "E": "1", "T": "2", "F": "3", "(": "S4", "id": "S5" },
        "I1": { "$": "acc", "+": "S6", "-": "S7" },
```

```
        "I2": { "$": "R3", "-": "R3", "+": "R3", "*": "S8",
"/": "S9" },
        "I3": { "/": "R6", "*": "R6", "+": "R6", "-": "R6",
"$": "R6" },
        "I4": { "E": "10", "T": "11", "F": "12", "(": "S13",
"id": "S14" },
        "I5": { "/": "R8", "*": "R8", "+": "R8", "$": "R8", "-
": "R8" },
        "I6": { "T": "15", "F": "3", "(": "S4", "id": "S5" },
        "I7": { "T": "16", "F": "3", "(": "S4", "id": "S5" },
        "I8": { "F": "17", "(": "S4", "id": "S5" },
        "I9": { "F": "18", "(": "S4", "id": "S5" },
        "I10": { ")": "S19", "+": "S20", "-": "S21" },
        "I11": { ")": "R3", "-": "R3", "+": "R3", "*": "S22",
"/": "S23" },
        "I12": { "/": "R6", "*": "R6", "+": "R6", ")": "R6", "-
": "R6" },
        "I13": { "E": "24", "T": "11", "F": "12", "(": "S13",
"id": "S14" },
        "I14": { "/": "R8", "*": "R8", "+": "R8", ")": "R8", "-
": "R8" },
        "I15": { "$": "R1", "-": "R1", "+": "R1", "*": "S8",
"/": "S9" },
        "I16": { "$": "R2", "-": "R2", "+": "R2", "*": "S8",
"/": "S9" },
        "I17": { "/": "R4", "*": "R4", "+": "R4", "-": "R4",
"$": "R4" },
        "I18": { "/": "R5", "*": "R5", "+": "R5", "-": "R5",
"$": "R5" },
        "I19": { "/": "R7", "*": "R7", "+": "R7", "$": "R7", "-
": "R7" },
        "I20": { "T": "25", "F": "12", "(": "S13", "id": "S14"
},
        "I21": { "T": "26", "F": "12", "(": "S13", "id": "S14"
},
        "I22": { "F": "27", "(": "S13", "id": "S14" },
        "I23": { "F": "28", "(": "S13", "id": "S14" },
        "I24": { ")": "S29", "+": "S20", "-": "S21" },
        "I25": { ")": "R1", "-": "R1", "+": "R1", "*": "S22",
"/": "S23" },
        "I26": { ")": "R2", "-": "R2", "+": "R2", "*": "S22",
"/": "S23" },
```

```
        "I27": { "/": "R4", "*": "R4", "+": "R4", ")": "R4", "-": "R4" },
        "I28": { "/": "R5", "*": "R5", "+": "R5", ")": "R5", "-": "R5" },
        "I29": { "/": "R7", "*": "R7", "+": "R7", ")": "R7", "-": "R7" }
    }
    '''
    return json.loads(talbe_json)


def lexical_sentence(sentence: str):
    # 分隔输入，只支持算数表达式
    input: list = list(re.findall(r"[0-9]+|[+\-*/()$]", sentence))
    lexical_words = []
    for item in input:
        item: str
        if item.isdigit():
            lexical_words.append(("id", int(item)))
        else:
            lexical_words.append((item, ''))
    return lexical_words


def LR_analysis(lexical_words: list, table: dict, grammar="", begin="E"):
    lexical_words.append(("$", ''))  # 放入终结符
    lexical_words.reverse()  # 倒置为栈
    # 初始化栈
    stack = ['0']       # 状态栈
    val_stack = ['']    # 计算值的栈

    while True:
        status = 'I' + stack[-1]
        item = lexical_words[-1]

        # 错误处理
        if item[0] not in table[status]:
            lexical_words.pop()
            print(f"error happend: drop {item}")
            continue

```

```
        # 结束条件
        action = table[status][item[0]]
        if action == "acc":
            print("accept, anylasis success!")
            eval(grammar['0'][2])  # 执行R0对应的翻译动作
            break

        # 移进
        op, label = action[0], action[1:]
        if op == "S":
            stack.append(item[0])
            stack.append(label)

            val_stack.append(item[1])
            val_stack.append('')
            lexical_words.pop()
            print(f"移进{item}")
        # 归约
        elif op == "R":
            index = label
            left, right = grammar[index][0], grammar[index][1]
            value = eval(grammar[index][2])  # 执行翻译动作

            # 弹出旧字符
            for symbal in reversed(right):
                while symbal is not stack[-1]:
                    stack.pop()
                    val_stack.pop()
                stack.pop()
                val_stack.pop()

            status = 'I' + stack[-1]
            goto = table[status][left]

            stack.append(left)
            stack.append(goto)
            val_stack.append(value)
            val_stack.append('')
            print(f"按照{left} -> {right}进行归约")
    pass


def main():
```

```
125     grammar = {
126         r'0': (r'L', [r'E'], r"print(f'表达式的值为{val_stack[-2]}')"),
127         r'1': (r'E', [r'E', '+', 'T'], r"val_stack[-6] + val_stack[-2]"),
128         r'2': (r'E', [r'E', '-', 'T'], r"val_stack[-6] - val_stack[-2]"),
129         r'3': (r'E', [r'T'], r"val_stack[-2]"),
130         r'4': (r'T', [r'T', '*', 'F'], r"val_stack[-6] * val_stack[-2]"),
131         r'5': (r'T', [r'T', '/', 'F'], r"val_stack[-6] / val_stack[-2]"),
132         r'6': (r'T', [r'F'], r"val_stack[-2]"),
133         r'7': (r'F', [r'(', 'E', ')'], r"val_stack[-4]"),
134         r'8': (r'F', [r'id'], r"val_stack[-2]"),
135     }
136     sentence = "4**2+4"  # 对应的表达式
137     print("请输入计算的表达式(直接敲击回车默认使用 4**2+4):", end='')
138     input_sentenct = input()
139     if input_sentenct != '':
140         sentence = input_sentenct
141     lexical_words = lexical_sentence(sentence=sentence)
142     table = get_table()
143     LR_analysis(lexical_words=lexical_words, table=table, grammar=grammar, begin='L')
144 
145     pass
146 
147 
148 if __name__ == "__main__":
149     main()
150 
```