الجلسة السادسة: اختبار

الهدف من الجلسة

اختبار المترجم عند وجود كود أكثرمن سطر.

ليكن لديك الكود التالي:

```
import ply.lex as lex
import ply.yacc as yacc

# ====================== SCANNER (LEX) ======================

# Token list
tokens = (
    'IF', 'THEN', 'ELSE',
    'ID', 'NUM',
    'LPAR', 'RPAR', 'SEMI',
    'LT', 'GT', 'LE', 'GE', 'EQ', 'NE',
    'OR', 'AND',
    'PLUS', 'MINUS', 'MULT', 'DIVS',
    'EQUAL'
)

# Reserved keywords
reserved = {
    'if': 'IF',
    'then': 'THEN',
    'else': 'ELSE',
}

# Regular expression rules for simple tokens
t_LPAR = r'\('
```

```
t_RPAR = r'\)'
t_SEMI = r';'
t_LT = r'<'
t_GT = r'>'
t_LE = r'<='
t_GE = r'>='
t_EQ = r'=='
t_NE = r'!='
t_OR = r'\|\|'
t_AND = r'&&'
t_PLUS = r'\+'
t_MINUS = r'-'
t_MULT = r'\*'
t_DIVS = r'/'
t_EQUAL = r'='

# Ignored characters (spaces and tabs)
t_ignore = ' \t'

# Regular expression rule for numbers
def t_NUM(t):
    r'\d+'
    t.value = int(t.value)
    return t

# Regular expression rule for identifiers and keywords
```

```
def t_ID(t):
    r'[a-zA-Z_][a-zA-Z_0-9]*'
    t.type = reserved.get(t.value, 'ID')  # Check for reserved words
    return t

# Track line numbers
def t_newline(t):
    r'\n+'
    t.lexer.lineno += len(t.value)

# Error handling
def t_error(t):
    print(f"Illegal character '{t.value[0]}'")
    t.lexer.skip(1)

# Build the lexer
lexer = lex.lex()

# ====================== PARSER (YACC) ======================

# Precedence rules (lowest to highest)
precedence = (
    ('right', 'EQUAL'),
    ('left', 'OR'),
    ('left', 'AND'),
    ('left', 'LT', 'GT', 'LE', 'GE', 'EQ', 'NE'),
```

```
    ('left', 'PLUS', 'MINUS'),
    ('left', 'MULT', 'DIVS'),
)

# Grammar rules
def p_S(p):
    "'S : ST'"
    print("Input accepted.")
    p[0] = p[1]

def p_ST_if_then_else(p):
    'ST : IF LPAR E2 RPAR THEN ST1 SEMI ELSE ST1 SEMI'
    p[0] = ('if_then_else', p[3], p[6], p[9])

def p_ST_if_then(p):
    'ST : IF LPAR E2 RPAR THEN ST1 SEMI'
    p[0] = ('if_then', p[3], p[6])

def p_ST1_statement(p):
    'ST1 : ST'
    p[0] = p[1]

def p_ST1_expression(p):
    'ST1 : E'
    p[0] = p[1]
```

```
# Expression rules for E (assignment and arithmetic)
def p_E_assignment(p):
    'E : ID EQUAL E'
    p[0] = ('assign', p[1], p[3]) #semantic action

def p_E_binop(p):
    '''E : E PLUS E
         | E MINUS E
         | E MULT E
         | E DIVS E
         | E LT E
         | E GT E
         | E LE E
         | E GE E
         | E EQ E
         | E NE E
         | E OR E
         | E AND E'''
    p[0] = ('binop', p[2], p[1], p[3]) #binop is a binary operation

def p_E_negative(p):
    'E : MINUS E'
    p[0] = ('neg', p[2])

def p_E_id(p):
    'E : ID'
```

```
    p[0] = ('id', p[1])

def p_E_num(p):
    'E : NUM'
    p[0] = ('num', p[1])

# Expression rules for E2 (conditional expressions only)
def p_E2_binop(p):
    '''E2 : E LT E
          | E GT E
          | E LE E
          | E GE E
          | E EQ E
          | E NE E
          | E OR E
          | E AND E'''
    p[0] = ('binop', p[2], p[1], p[3])

def p_E2_id(p):
    'E2 : ID'
    p[0] = ('id', p[1])

def p_E2_num(p):
    'E2 : NUM'
    p[0] = ('num', p[1])
```

```
def p_error(p):
    if p:
        print(f"Syntax error at '{p.value}'")
    else:
        print("Syntax error at EOF")

# Build the parser
parser = yacc.yacc()

# ==================== FILE READING & MAIN EXECUTION ====================

def read_from_file(filename):
    """Read input from file"""
    try:
        with open(filename, 'r') as f:
            return f.read()
    except FileNotFoundError:
        print(f"Error: {filename} not found.")
        return None
    except Exception as e:
        print(f"Error reading file: {e}")
        return None

def main():
    print("IF-THEN-ELSE Compiler")
```

```
    print("=" * 30)

    # Read from input.txt file
    input_data = read_from_file('input.txt')
    print(f"Reading from input.txt:")
    print(f"Content: {input_data}")
    tests=input_data.split("\n")
    print("-" * 30)
    for t in tests:
        result = parser.parse(t)
        print(f"Parse result: {result}")

if __name__ == "__main__":
    main()
```

المطلوب:

1. أنشيء مجلد جديد في الـvisual studio code باسم compiler_code
2. أنشيء ملف نصي باسم input ضمن نفس مجلد العمل واكتب ضمنه النص التالي:

```
int  x=1;

If(x>10) then print("hello");

bool y=x||10;
```

3. عدل الكود السابق بحيث يقبل ترجمة الكود الموجود في الملف input.txt.
4. اجعل المعرب يقبل عبارة if المتداخلة في الاستعلام.

الحل:

1. التعديلات اللازمة:

في الـ tokens:

```
tokens = (
    'IF', 'THEN', 'ELSE',
    'ID', 'NUM',
    'LPAR', 'RPAR', 'SEMI',
    'LT', 'GT', 'LE', 'GE', 'EQ', 'NE',
    'OR', 'AND','INT','BOOL',
    'PLUS', 'MINUS', 'MULT', 'DIVS',
    'EQUAL','PRINT','DQ','TRUE', 'FALSE', 'INC','DEC',
    'RCPAR','LCPAR'
)
```

الـ reserved words:

```
reserved = {
    'if': 'IF',
    'then': 'THEN',
    'else': 'ELSE',
    'int': 'INT',
    'bool': 'BOOL',
    'char': 'CHAR',
    'print': 'PRINT',
    'true' : 'TRUE',
    'false' :'FALSE'
}
```

```
t_LPAR = r'\('
t_RPAR = r'\)'
t_SEMI = r';'
t_LT = r'<'
t_GT = r'>'
t_LE = r'<='
t_GE = r'>='
t_EQ = r'=='
t_NE = r'!='
t_OR = r'\|\|'
t_AND = r'&&'
t_INC=r'\+\+'
t_DEC=r'\-\-'
t_PLUS = r'\+'
t_MINUS = r'-'
t_MULT = r'\*'
t_DIVS = r'/'
t_EQUAL = r'='
t_DQ= r'\"'
t_RCPAR='\{'
t_LCPAR='\}'
t_ignore = ' \t'
```

القواعد اللازمة للتعرف على تعريف متغيرات وإسناد قيم لها:

```
def p_ST_exp_assign(p):
    'ST : TYPE ID EQUAL E4 SEMI'
    p[0]=('assign',p[2],p[4])

def p_ST_datatype(p):
    '''TYPE : INT
            | BOOL
    '''
    p[0]=('datatype',p[1])

def p_E4(p):
    'E4 : ST1'
    p[0]=('assign statement',p[1])
```

القواعد اللازمة لإعراب السطر الخاص بالشرط:

```
def p_ST1_print(p):
    'ST1 : PRINT LPAR DQ E3 DQ RPAR'
    p[0] = ('print st',p[1],p[4])
def p_E3(p):
    'E3 : E'
    p[0] = ('string',p[1])
```

5. الحل: if(x>3) then if(y>4) then print("z");;

انتهت الجلسة

إعداد:م. ريم جبيلي