Міністерство освіти і науки України
Львівський національний університет імені Івана Франка Факультет електроніки та комп’ютерних технологій

Звіт

про виконання лабораторної роботи №8

з курсу “Функціональне програмування ”

“**Алгебраїчні типи даних “поPython’івськи”**”

Виконав:
студент групи ФЕП-23
Чепара Станіслав
Перевірив:
доцент Франів В. А.

Львів 2025

Мета роботи: Union, Literal, Protocol, структурний pattern matching (match/case) для розбору структур.

Хід роботи

1. модель виразів Expr = Num | Add | Mul

expr.py

expr.py > SupportsBin

```python
from __future__ import annotations
from dataclasses import dataclass
from typing import TypeAlias, Literal, Protocol, Union

@dataclass(frozen=True)
class Num:
    value: int

@dataclass(frozen=True)
class Add:
    left: "Expr"
    right: "Expr"

@dataclass(frozen=True)
class Mul:
    left: "Expr"
    right: "Expr"

# Type alias (Python 3.10-3.11 style; on 3.12+ you can use: `type Expr = ...`)
Expr: TypeAlias = Num | Add | Mul

# Optional: enable positional matching (case Add(l, r)) in addition to named fields
Num.__match_args__ = ("value",)
Add.__match_args__ = ("left", "right")
Mul.__match_args__ = ("left", "right")

# --- Alternative AST using Literal and a single binary node ----------------
Op: TypeAlias = Literal["+", "*"]

@dataclass(frozen=True)
class Bin:
    op: Op
    left: "Expr2"
    right: "Expr2"

@dataclass(frozen=True)
class Num2:
    value: int

Expr2: TypeAlias = Union[Num2, Bin]

class SupportsBin(Protocol):
    op: Op
    left: "Expr2"
    right: "Expr2"
```

2. «чистий» eval через match

eval_.py

eval_.py > ...

```python
from __future__ import annotations
from typing import Final
from expr import Expr, Num, Add, Mul, Expr2, Num2, Bin

# Simple overflow guard (demo of an invariant)
MAX_ABS: Final[int] = 10_000

class EvalOverflow(ValueError):
    pass

def _check_range(x: int) -> int:
    if abs(x) > MAX_ABS:
        raise EvalOverflow(f"overflow: {x}")
    return x

def eval_expr(e: "Expr") -> int:
    match e:
        case Num(value=v):
            return _check_range(v)
        case Add(left=a, right=b):
            return _check_range(eval_expr(a) + eval_expr(b))
        case Mul(left=a, right=b):
            return _check_range(eval_expr(a) * eval_expr(b))

# Alternative evaluator for the Literal/Op variant
def eval_expr2(e: "Expr2") -> int:
    match e:
        case Num2(value=v):
            return _check_range(v)
        case Bin(op="+", left=a, right=b):
            return _check_range(eval_expr2(a) + eval_expr2(b))
        case Bin(op="*", left=a, right=b):
            return _check_range(eval_expr2(a) * eval_expr2(b))

```

Результати:

```
(.venv) stanislav@fedora:~/Desktop/Functional_Programing/lab08$ python main.py
AST: ((2 + 3) * 4)
eval: 20
AST2: ((2 + 3) * 4)
eval2: 20
(.venv) stanislav@fedora:~/Desktop/Functional_Programing/lab08$ pytest -v
============================================================================= test session starts
platform linux -- Python 3.13.9, pytest-8.4.2, pluggy-1.6.0
rootdir: /home/stanislav/Desktop/Functional_Programing/lab08
configfile: pyproject.toml
testpaths: tests
plugins: anyio-4.9.0
collected 5 items

tests/test_expr.py .....

============================================================================== 5 passed in 0.04s =
(.venv) stanislav@fedora:~/Desktop/Functional_Programing/lab08$
```

Висновок: розібрався із створенням моделі математичних виразів (AST) та їх обчислювач за допомогою класів даних і структурного зіставлення зразків (match/case).