1 Удирдамж

Зорилго: Интерпретаторын үндсийг ойлгох, хялбар туршилт хийх.

Тодорхойлолт: Энэхүү лабораторийн ажил нь энгийн арифметик илэрхийлэл шинжлэгчийн кодыг хөгжүүлэхэд чиглэнэ. Даалгаврууд нь программчлалын хэлийг сонгох, үндсэн эхлэл функцийн кодыг бэлтгэх, интерпретаторын бүтцийг тодорхойлох, үндсэн токен задаргааг хэрэгжүүлэх, арифметик үйлдлүүдэд (нэмэх, хасах гэх мэт) оруулах, хялбар арифметик илэрхийллүүд дээр тест хийх зэрэг орно.

Зөвлөмж: Интерпретаторыг системтэйгээр хөгжүүлэхийн тулд даалгавар бүрийг дараалан гүйцэтгээрэй. Тестийн үе шат нь интерпретаторын компонентуудын ажиллагааг баталгаажуулахад маш чухал. Даалгаврууд ыг илүү сайн ойлгохын тулд псевдо кодын жишээг ашиглана уу. Энэхүү лабораторийн ажил нь дараа долоо хоногт судлах хэлний синтаксын илүү ярвигтай хэсэгт суурь тавих практик туршлага болно.

2 Гүйцэтгэх даалгавар

Алхам 1: Бэлтгэл

- Интерпретатор хөгжүүлэх программчлалын хэлээ сонгох: C++, Python.
- Эхлэл main функцийн үндсэн кодыг бичих.

```
// Interpreter
function main():
    expression = input("Enter an arithmetic expression: ")
    result = interpret(expression)
    print("Result:", result)

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

Санамж: Сонгосон программчлалын хэлтэй танилцаж, таны хөгжүүлэлтийн орчин бэлэн байгаа эсэхийг шалгаарай.

Алхам 2: Интерпретаторын үндсэн хэсэг

- Интерпретаторын бүтцийг тодорхойлох (задлах, хөрвүүлэх функцүүд).
- Арифметик илэрхийлэлд зориулан үндсэн токенуудаар задлах үйлдлийг хэрэгжүүлэх.

```
// Interpreter
function interpret(expression):
    tokens = tokenize(expression)
    // Further interpretation logic goes here

function tokenize(expression):
    // Simple tokenization for illustration purposes
    return split(replace(expression, '+', ' + '), ' ')
```

Санамж: Интерпретаторыг хялбар удирдагдах функцүүдэд жижиглэж хуваагаарай. Интерпретатор илэрхийллийг хэрхэн ойлгон задалж буйг нарийн мэдэх хэрэгтэй.

Алхам 3: Арифметик үйлдлүүдийн хэсэг

- Нэмэх, хасах үйлдэлтэй байна.
- Өөр өөр арифметик илэрхийлэл дээр тест хийнэ.

```
// Interpreter
function interpret(expression):
    tokens = tokenize(expression)
    result = toInteger(tokens[0])  // Initial value
    for i = 1 to length(tokens) step 2:
        operator = tokens[i]
        operand = toInteger(tokens[i + 1])
        if operator == '+':
            result += operand
        else if operator == '-':
            result -= operand
    return result
```

Санамж: Үржүүлэх, хуваах болон илүү төвөгтэй илэрхийллийг хэрхэн зохицуулах талаар бодож үзээрэй. Илэрхийллийг илүү сайн шинжлэхийн тулд стек ашиглана уу.

Алхам 4: Тестчилэл

- Туршилтын иж бүрдлийг боловсруулна.
- Интерпретаторыг өөр өөр илэрхийллээр туршина.

```
// TestCases
import unittest
from Interpreter import interpret

class TestInterpreter(unittest.TestCase):
    function test_addition():
        assertEqual(interpret("3 + 5"), 8)

    function test_subtraction():
        assertEqual(interpret("7 - 2"), 5)

    function test_complex_expression():
        assertEqual(interpret("2 * (4 + 3)"), 14)

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

Санамж: Туршилтын тохиолдлуудад захын тохиолдол, сөрөг тоо, хашилтыг анхаарах хэрэгтэй. Туршилтын шаталсан хэлбэрийг ашиглан аливаа асуудлыг тодорхойлж, засах байдлаар хөгжүүлэх.

ТАНД АМЖИЛТ ХҮСЬЕ!