НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ

Факультет інформаційних технологій

Кафедра комп’ютерних наук

Теорія розпізнавання образів та класифікації в системах штучного інтелекту

Лабораторна робота №11

Виконав:

Студент групи КН-20002Б

Кропивка Анатолій Анатолійович

Київ 2023

Тема: Дослідження методів синтаксичного аналізу в мовних процесорах.

Мета: Дослідження методів побудови синтаксичних аналізаторів.

Підготовка до роботи: Вивчити й уяснити призначення і зміст завдання до лабораторної роботи.

Хід роботи:

1. Ознайомитись з методичною розробкою до лабораторної роботи.
2. Ознайомитись з рекомендованною літературою.
3. Розробити для проведення дослідження програмний застосунок синтаксичного аналізатора на мові програмування C++.
4. Вибрати і описати у звіті відповідні хеш-функції.
5. Виконати дослідження у відповідності з завданням до лабораторної роботи.
6. За результатами досліджень скласти звіт з обґрунтованими висновками.

Опис програм

Програма є калькулятором, який обчислює значення арифметичних виразів, введених користувачем. Вона розбиває вираз на складники *(терми)* та обчислює їх значення, використовуючи бінарні операції *(+, -, *, /)*. Кожен складник може бути числом або іншим виразом, який також підлягає обчисленню. Програма використовує класи *Node*, *NumberNode* та *BinaryOpNode*, які представляють вузли *(елементи)* абстрактного синтаксичного дерева *(AST)*, імплементують метод *evaluate*, який обчислює значення вузла.

Програма також містить *NodeBuilder*, який відповідає за розбір введеного виразу та побудову *AST*. Він проходить по виразу, використовуючи методи *parseExpression*, *parseTerm* та *parseFactor*, щоб створити відповідні вузли дерева.

У основному циклі програми користувач може вводити вирази для обчислення. Програма також має додатковий потік *task*, який виконується паралельно з основним циклом програми. Цей потік слугує для перевірки натискання клавіші *esc*, яка призводить до виходу з програми.

Код програми

```
#include <Windows.h>
#include <chrono>
#include <cstdlib>
#include <iostream>
#include <memory>
#include <string>
#include <thread>

using namespace std;

string getString()
{
   string value {};

   try {
      getline(cin, value);
   } catch (const exception& e) {
      cout << "Failed to read a line: " << e.what() << endl;
   }

   return value;
}

class Node {
public:
   virtual int evaluate() const = 0;
};

class NumberNode : public Node {
public:
```

```
    NumberNode(int value)
        : value(value)
    {
    }

    int evaluate() const override
    {
        return value;
    }

private:
    int value;
};

class BinaryOpNode : public Node {
public:
    BinaryOpNode(char op, unique_ptr<Node> left, unique_ptr<Node> right)
        : op(op)
        , left(move(left))
        , right(move(right))
    {
    }

    int evaluate() const override
    {
        switch (op) {
        case '+':
            return left->evaluate() + right->evaluate();
        case '-':
            return left->evaluate() - right->evaluate();
        case '*':
            return left->evaluate() * right->evaluate();
```

```
        case '/':
            return left->evaluate() / right->evaluate();
        default:
            throw runtime_error("Invalid operator");
        }
    }

private:
    char op;
    unique_ptr<Node> left;
    unique_ptr<Node> right;
};

class NodeBuilder {
public:
    explicit NodeBuilder(const string& expression)
        : expression(expression)
        , index(0)
    {
    }

    unique_ptr<Node> buildAST()
    {
        return parseExpression();
    }

private:
    unique_ptr<Node> parseExpression()
    {
        auto left = parseTerm();
        while (match('+') || match('-')) {
            char op = expression[index - 1];
```

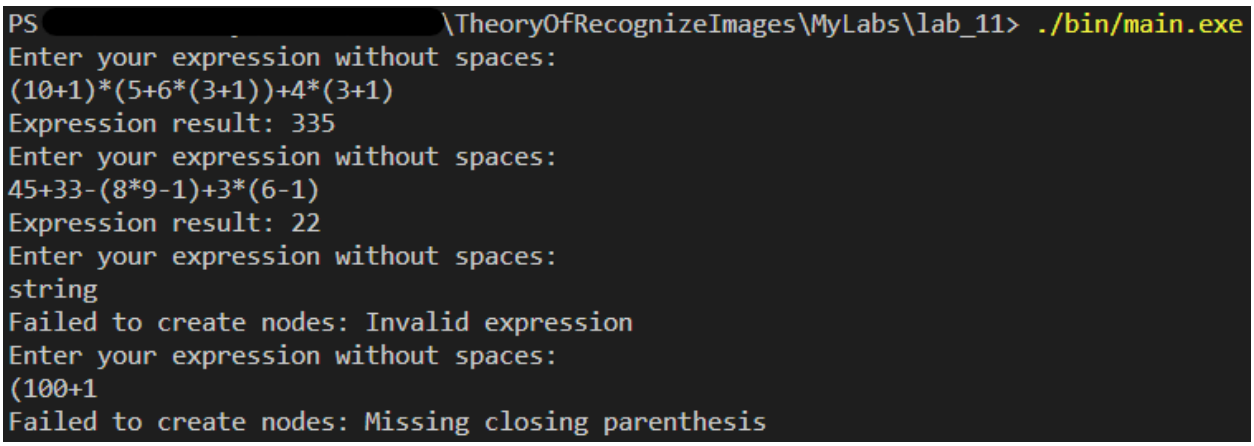
```
        auto right = parseTerm();
        left = make_unique<BinaryOpNode>(op, move(left), move(right));
    }
    return left;
}

unique_ptr<Node> parseTerm()
{
    auto left = parseFactor();
    while (match('*') || match('/')) {
        char op = expression[index - 1];
        auto right = parseFactor();
        left = make_unique<BinaryOpNode>(op, move(left), move(right));
    }
    return left;
}

unique_ptr<Node> parseFactor()
{
    if (isdigit(peek())) {
        int value = parseNumber();
        return make_unique<NumberNode>(value);
    } else if (match('(')) {
        auto node = parseExpression();
        if (!match(')')) {
            throw runtime_error("Missing closing parenthesis");
        }
        return node;
    } else {
        throw runtime_error("Invalid expression");
    }
}
```

```
int parseNumber()
{
    string number;
    while (isdigit(peek())) {
        number += advance();
    }
    return stoi(number);
}

char peek() const
{
    if (index < expression.size()) {
        return expression[index];
    }
    return '\0';
}

char advance()
{
    if (index < expression.size()) {
        return expression[index++];
    }
    return '\0';
}

bool match(char expected)
{
    if (peek() == expected) {
        advance();
        return true;
    }
```

```
        return false;
    }

    string expression;
    size_t index;
};

int main()
{
    thread task([]() {
        while (true) {
            this_thread::sleep_for(chrono::milliseconds(100));

            if (GetAsyncKeyState(VK_ESCAPE)) {
                exit(0);
            }
        }
    });

    while (true) {
        cout << "Enter your expression without spaces: " << endl;
        string expression { getString() };
        NodeBuilder builder(expression);

        try {
            unique_ptr<Node> ast = builder.buildAST();
            int result = ast->evaluate();
            cout << "Expression result: " << result << endl;
        } catch (const exception& e) {
            cout << "Failed to create nodes: " << e.what() << endl;
        }
    }
```

```
    return 0;
}
```

Знімки екрану



Висновок: Дана програма є калькулятором, який може обчислювати значення арифметичних виразів. Вона використовує абстрактне синтаксичне дерево *(AST)* для представлення виразу та обчислення його значення. Програма демонструє використання класів, виключень, роботу зі стрічками та роботу зі стандартним вводом/виводом.