МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ
УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ «ПОЛОЦКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИМЕНИ ЕВФРОСИНИИ ПОЛОЦКОЙ»

«Компиляторные технологии»
Отчет по лабораторной работе №4
«Генерация и оптимизация объектного кода»

Выполнил: Студент группы _____
ФИО

Проверил: Преподаватель
Пяткин Д.В.

Полоцк, 2024

ЛАБОРАТОРНАЯ РАБОТА №4

Цель работы: изучение основных принципов генерации компилятором объектного кода, ознакомление с методами оптимизации результирующего объектного кода для линейного участка программы с помощью свертки и исключения лишних операций.

1. Ход выполнения лабораторной работы

Вариант №1.

Написать программу, которая выполняет лексический анализ входного текста в соответствии с заданием, порождает таблицу лексем и выполняет синтаксический разбор текста по заданной грамматике, построить генератор триад, который выводит список триад до оптимизации и после.

Вариант грамматики:

1. S → a:= F;
F → F+T |T
T → T·E | TIE | E
E → (F) | – (F) | a

Выполненная программа представлена в листинге 1.

Листинг 1 – Программа, соответствующая заданию.

```
using System;
using System.Collections.Generic;

public enum TokenType
{
    EQ,
    SEMI,
    LPAREN,
    RPAREN,
    ADD,
    SUB,
    MUL,
    DIV,
    NUM,
    ID,
    END
}

public class Token
{
    public TokenType type;
    public string value;

    public Token(TokenType type, string value)
    {
```

```
        this.type = type;
        this.value = value;
    }
}

public class Triad
{
    public string Operation { get; set; }
    public string Operand1 { get; set; }
    public string Operand2 { get; set; }

    public Triad(string operation, string operand1, string operand2)
    {
        Operation = operation;
        Operand1 = operand1;
        Operand2 = operand2;
    }

    public override string ToString()
    {
        return $"{Operation} ({Operand1}, {Operand2})";
    }
}

public class Parser
{
    private string input;
    private int position;
    private List<Token> tokens = new List<Token>();
    private int currentToken = 0;
    private List<Triad> triads = new List<Triad>();
    private Dictionary<string, int> expressionToTriadIndex = new Dictionary<string, int>();

    public Parser(string input)
    {
        this.input = input;
        this.position = 0;
        TokenStream();
    }

    private void TokenStream()
    {
        while (position < input.Length)
        {
            while (position < input.Length && char.IsWhiteSpace(input[position]))
            {
                position++;
            }

            if (position >= input.Length)
            {
                break;
            }

            if (char.IsDigit(input[position]) || input[position] == '.')
            {
```

```
            string num = "";
            while (position < input.Length && (char.IsDigit(input[position]) || input[position] == '.'))
            {
                num += input[position++];
            }
            tokens.Add(new Token(TokenType.NUM, num));
        }
        else
        {
            char ch = input[position];
            switch (ch)
            {
                case ':':
                    if (position + 1 < input.Length && input[position + 1] == '=')
                    {
                        tokens.Add(new Token(TokenType.EQ, ":="));
                        position += 2;
                    }
                    else
                    {
                        throw new Exception("Unexpected character: " + ch);
                    }
                    break;
                case ';':
                    tokens.Add(new Token(TokenType.SEMI, ";"));
                    position++;
                    break;
                case '(':
                    tokens.Add(new Token(TokenType.LPAREN, "("));
                    position++;
                    break;
                case ')':
                    tokens.Add(new Token(TokenType.RPAREN, ")"));
                    position++;
                    break;
                case '+':
                    tokens.Add(new Token(TokenType.ADD, "+"));
                    position++;
                    break;
                case '-':
                    tokens.Add(new Token(TokenType.SUB, "-"));
                    position++;
                    break;
                case '*':
                    tokens.Add(new Token(TokenType.MUL, "*"));
                    position++;
                    break;
                case '/':
                    tokens.Add(new Token(TokenType.DIV, "/"));
                    position++;
                    break;
                default:
                    if (char.IsLetter(ch) || ch == '_')
                    {
                        string id = "";
                        while (position < input.Length && (char.IsLetterOrDigit(input[position]) || input[position] == '_'))
```

```
                {
                    id += input[position++];
                }
                tokens.Add(new Token(TokenType.ID, id));
            }
            else
            {
                throw new Exception("Invalid character: " + ch);
            }
            break;
        }
    }
  }
  tokens.Add(new Token(TokenType.END, ""));
}

private string GetOrAddTriad(string op, string left, string right)
{
  string key = $"{op},{left},{right}";
  if (expressionToTriadIndex.TryGetValue(key, out int triadIndex))
  {
    return $"^{triadIndex + 1}";
  }
  else
  {
    triads.Add(new Triad(op, left, right));
    int newTriadIndex = triads.Count - 1;
    expressionToTriadIndex[key] = newTriadIndex;
    return $"^{newTriadIndex + 1}";
  }
}


public string ParseE()
{
  string res = ParseT();
  while (currentToken < tokens.Count &&
      (tokens[currentToken].type == TokenType.ADD || tokens[currentToken].type == TokenType.SUB))
  {
    string op = tokens[currentToken].value;
    currentToken++;
    string right = ParseT();
    res = GetOrAddTriad(op, res, right);
  }
  return res;
}

public string ParseT()
{
  string res = ParseF();
  while (currentToken < tokens.Count &&
      (tokens[currentToken].type == TokenType.MUL || tokens[currentToken].type == TokenType.DIV))
  {
    string op = tokens[currentToken].value;
    currentToken++;
    string right = ParseF();
```

```
            res = GetOrAddTriad(op, res, right);
        }
        return res;
    }

    public string ParseF()
    {
        if (tokens[currentToken].type == TokenType.NUM || tokens[currentToken].type == TokenType.ID)
        {
            string value = tokens[currentToken].value;
            currentToken++;
            return value;
        }
        else if (tokens[currentToken].type == TokenType.LPAREN)
        {
            currentToken++;
            string res = ParseE();
            if (tokens[currentToken].type != TokenType.RPAREN)
                throw new Exception("Expected ')'");
            currentToken++;
            return res;
        }
        else if (tokens[currentToken].type == TokenType.SUB)
        {
            currentToken++;
            string right = ParseF();
            return GetOrAddTriad("-", "^0", right);
        }
        else
        {
            throw new Exception("Expected NUM, ID, or '('");
        }
    }

    public void ParseS()
    {
        string left = tokens[currentToken].value;
        currentToken += 2;
        string right = ParseE();
        triads.Add(new Triad(":=", left, right));

        if (tokens[currentToken].type != TokenType.SEMI)
            throw new Exception("Expected ';'");
        currentToken++;
    }

    public void PrintTriads(List<Triad> triads)
    {
        foreach (Triad triad in triads)
        {
            Console.WriteLine(triad);
        }
    }
    public List<Triad> OptimizeTriads(List<Triad> triads)
    {
        var optimizedTriads = new List<Triad>();
```

```
    var triadSet = new HashSet<string>();

    foreach (var triad in triads)
    {
        string triadString = $"{triad.Operation},{triad.Operand1},{triad.Operand2}";

        if (triad.Operation == ":=" && triad.Operand1 == triad.Operand2)
        {
            continue;
        }

        if (triadSet.Contains(triadString))
        {
            continue;
        }

        optimizedTriads.Add(triad);
        triadSet.Add(triadString);
    }

    return optimizedTriads;
}

public void GenerateAndPrintTriads()
{
    try
    {
        while (currentToken < tokens.Count && tokens[currentToken].type != TokenType.END)
        {
            ParseS();
        }

        for (int i = 0; i < triads.Count; i++)
        {
            Triad triad = triads[i];
            if (int.TryParse(triad.Operand1.TrimStart('^'), out int index1))
            {
                triad.Operand1 = $"^{index1}";
            }
            if (int.TryParse(triad.Operand2.TrimStart('^'), out int index2))
            {
                triad.Operand2 = $"^{index2}";
            }
        }

        Console.WriteLine("Triads before optimization:");
        PrintTriads(triads);

        var optimizedTriads = OptimizeTriads(triads);

        Console.WriteLine("Triads after optimization:");
        PrintTriads(optimizedTriads);
    }
    catch (Exception e)
    {
        Console.WriteLine("Parsing error: " + e.Message);
```

```
        }
    }
}

public class Program
{
    public static void Main(string[] args)
    {
        string input = "x := 2.4 * y + - (z + a); x := 2.4 * y + - (z + a); x:= x;";
        Parser parser = new Parser(input);
        parser.GenerateAndPrintTriads();
    }
            }
```

Работа программы представлена на рисунке 1.

Рисунок 1 – Результат построения дерева вывода.

2. Контрольные вопросы

1. Что такое транслятор, компилятор и интерпретатор? Расскажите об общей структуре компилятора.
2. Как строится дерево вывода (синтаксического разбора)? Какие исходные данные необходимы для его построения?
3. Какую роль выполняет генерация объектного кода? Какие данные необходимы компилятору для генерации объектного кода? Какие действия выполняет компилятор перед генерацией?
4. Объясните, почему генерация объектного кода выполняется компилятором по отдельным синтаксическим конструкциям, а не для всей исходной программы в целом.
5. Расскажите, что такое синтаксически управляемый перевод.
6. Объясните работу алгоритма генерации последовательности триад по дереву синтаксического разбора на своем примере.
7. За счет чего обеспечивается возможность генерации кода на разных объектных языках по одному и тому же дереву?
8. Дайте определение понятию оптимизации программы. Для чего используется оптимизация? Каким условиям должна удовлетворять оптимизация?
9. Объясните, почему генерацию программы приходится проводить в два этапа: генерация и оптимизация.
10. Какие существуют методы оптимизации объектного кода?
11. Что такое триады и для чего они используются? Какие еще существуют методы для представления объектных команд?
12. Объясните работу алгоритма свертки. Приведите пример выполнения свертки объектного кода.
13. Что такое лишняя операция? Что такое число зависимости?
14. Объясните работу алгоритма исключения лишних операций. Приведите пример исключения лишних операций.

Ответы на вопросы

//////////////////////

//////////////////

//////////////////////

3. Вывод

В результате данной лабораторной работы, я изучил основные принципы генерации компилятором объектного кода, ознакомился с методами оптимизации результирующего объектного кода для линейного участка программы с помощью свертки и исключения лишних операций.