ГУАП

КАФЕДРА № 44

ОТЧЕТ
ЗАЩИЩЕН С ОЦЕНКОЙ

ПРЕПОДАВАТЕЛЬ

канд. техн. наук, доцент		Н.В. Кучин
должность, уч. степень, звание	подпись, дата	инициалы, фамилия

ОТЧЕТ ПО ЛАБОРАТОРНЫМ РАБОТАМ №6-7

ГЕНЕРАЦИЯ И ОПТИМИЗАЦИЯ ОБЪЕКТНОГО КОДА

по курсу: СИСТЕМНОЕ ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ

РАБОТУ ВЫПОЛНИЛ

СТУДЕНТ ГР. № 4941		Н. С. Горбунов
	подпись, дата	инициалы, фамилия

Санкт-Петербург 2022

1 Задание по лабораторной работе

Вариант 7

Входной язык содержит арифметические выражения, разделенные символом ; (точка с запятой). Арифметические выражения состоят из идентификаторов, римских чисел, знака присваивания (:=), знаков операций +, -, *, / и круглых скобок.

$$S \rightarrow \mathbf{a} := F;$$

$$F \rightarrow F + T \mid T$$

$$T \rightarrow T * E \mid T/E \mid E$$

$$E \rightarrow (F) \mid -(F) \mid \mathbf{a}$$

2 Цель работы

- Изучение основных принципов генерации компилятором объектного кода, выполнение генерации объектного кода программы на основе результатов синтаксического анализа для заданного входного языка.
- Изучение основных принципов оптимизации компилятором объектного кода для линейного участка программы, ознакомление с методами оптимизации результирующего объектного кода с помощью методов свертки объектного кода и исключения лишних операций.

3 Краткое описание лабораторной работы

Требуется написать программу, которая на основании дерева синтаксического разбора порождает объектный код. Программу рекомендуется построить из двух основных частей: первая часть – порождение дерева синтаксического разбора, вторая часть – реализация алгоритма порождения объектного кода по дереву разбора.

Результатами работы должна быть построенная на основании заданного предложения грамматики программа на объектном языке. В качестве объектного языка предлагается взять триады. Все встречающиеся в исходной программе идентификаторы считать простыми скалярными переменными, не требующими выполнения преобразования типов.

В данной работе алгоритм преобразования триад в команды языка ассемблера предлагается разработать самостоятельно. В тривиальном виде такой алгоритм заменяет каждую триаду на последовательность соответствующих команд, а результат ее выполнения запоминается во временной переменной с некоторым именем (например, TMPi, где i - номер триады). Тогда вместо ссылки на эту триаду в другой триаде будет подставлено значение этой переменной. Однако алгоритм может предусматривать и оптимизацию временных переменных.

4 Программная реализация

Была написана программная реализация генератора объектного кода, его оптимизации и генератора ассемблерного кода на C#.

Алгоритм работы метода генерации объектного кода в главном методе:

1. Получение исходного текста программы;
2. Формирование дерева вывода с синтаксическим и лексическим анализом;
3. Формирование триад из получившегося синтаксического дерева вывода;
4. Генерация ассемблерного кода из полученного списка триад;
5. Отображение результата обработки.

5 Полученные результаты

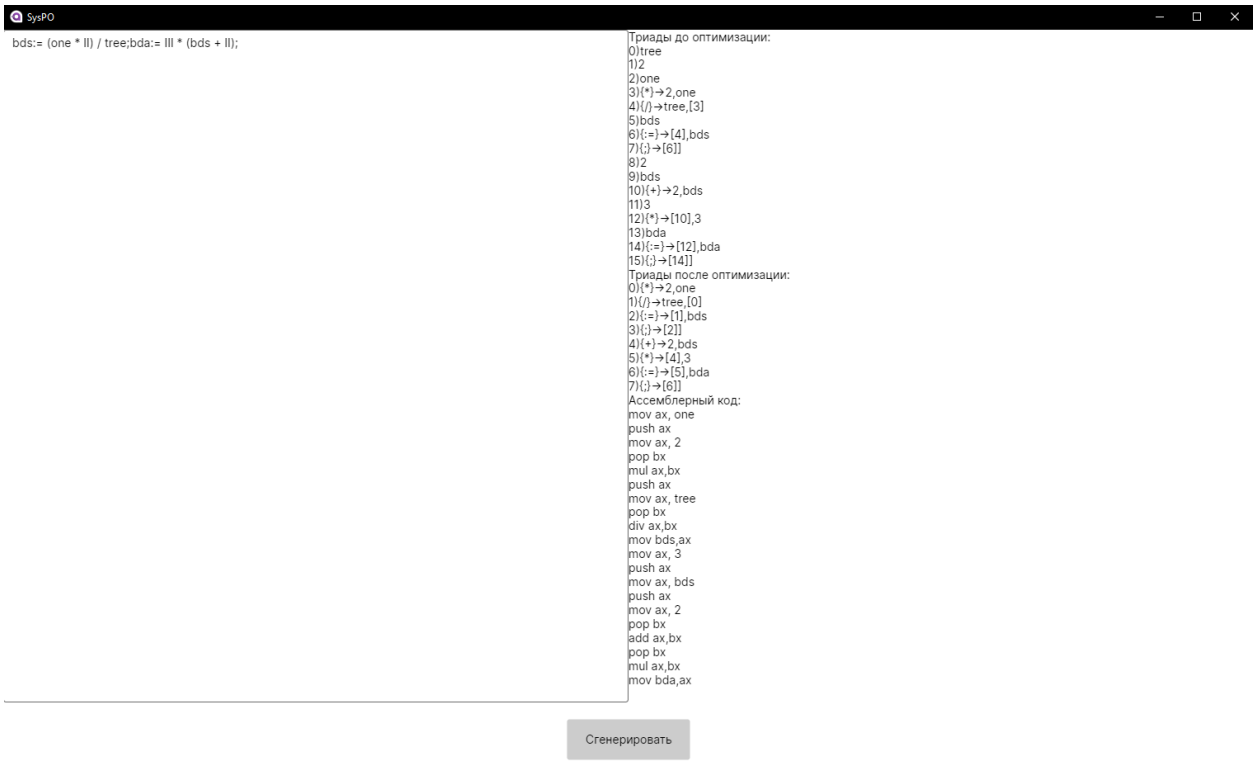

Рисунок 1 – Генерация ответа в программе

6 Вывод

В ходе работы были изучены основные принципы генерации компилятором объектного кода. Была выполнена генерации объектного кода программы на основе результатов синтаксического анализа для заданного входного языка.

Освоены основные принципы оптимизации компилятором объектного кода для линейного участка программы. Получены знания о методах оптимизации результирующего объектного кода с помощью методов свертки объектного кода и исключения лишних операций.

ПРИЛОЖЕНИЕ 1

Листинг 1. Модели триад

```
public interface ITriad
{
    int Id { get; set; }
}


public class BiTriad:ITriad // триада с двумя операндами
{
    public BiTriad(ITriad left, GrammarSymbol @operator, ITriad right)
    {
        Left = left;
        Operator = @operator;
        Right = right;
    }

    public ITriad Left { get; set; }
    public GrammarSymbol Operator{ get; init; }
    public ITriad Right { get; set; }
    public int Id { get; set;  }
    public override string ToString()
    {
        return $"{Operator.Symbol}->{(Left is SymbolTriad ? Left.ToString():
$"[{Left.Id}]" )}" +
                $",{(Right is SymbolTriad? Right.ToString():
$"[{Right.Id}]")}";
    }
}

public class UnoTriad : ITriad //триада с одним операндом
{
    public UnoTriad( ITriad operand, GrammarSymbol @operator)
    {
        Operand = operand;
        Operator = @operator;
    }

    public int Id { get; set; }
    public ITriad Operand { get; set; }
    public GrammarSymbol Operator { get; init; }
    public override string ToString()
    {
        return $"{Operator.Symbol}->[{(Operand is SymbolTriad ?
Operand.ToString(): Operand.Id)}]]";
    }
}

public class SymbolTriad : ITriad //заглушка. Триада означающая идентификатор
или число
{
    public SymbolTriad( GrammarSymbol symbol)
    {
        Symbol = symbol;
    }

    public int Id { get; set; }
    public GrammarSymbol Symbol { get; init; }
    public override string ToString()
    {
```

```
        return $"{Symbol.Word}";
    }
}

public static class TriadExtension
{
    //пронумеровать список триад для красивого вывода
    public static void Numerate(this List<ITriad> list)
    {
        for (int i = 0; i < list.Count; i++)
        {
            list[i].Id = i;
        }
    }
}
```

Листинг 2. Генерация объектного кода по дереву вывода

```
public class ObjectCodeGen
{
    public List<ITriad> _triadList { get; init; }

    public ObjectCodeGen()
    {
        _triadList = new List<ITriad>();
    }

    public ITriad GetTriad(Chain chain)
    {
    // начинаем разбор дерева вывода сверху
        ITriad? triad = null;
        switch (chain.Children?.Count) //по количеству листьев
        {
            case 3:
                if (chain.Children[1].Children != null &&
chain.Children[1].Children.Any())
                    return GetTriad(chain.Children[1]); //обработка
присваения
                triad = new BiTriad(GetTriad(chain.Children[0]),
chain.Children[1].Symbol, GetTriad(chain.Children[2])); //любая другая
операция с двумя операндами
                break;
            case 2:
                if (chain.Children[0].Symbol.IsTerminal) //унарный оператор
может находится и слева и справа
                    triad = new UnoTriad(GetTriad(chain.Children[1]),
chain.Children[0].Symbol);
                else
                    triad = new UnoTriad(GetTriad(chain.Children[0]),
chain.Children[1].Symbol);
                break;
            case 1:
                triad = GetTriad(chain.Children[0]);
                return triad;
            default:
                triad = new SymbolTriad(chain.Symbol);
                break;
        }
        _triadList.Add(triad);
        return triad;
    }
}
```

Листинг 3. Оптимизация объектного кода

```
public class Optimizer
{
    private ITriad[] _initial;
    private List<ITriad> _removable = new List<ITriad>();

        //оптимизация
    public Optimizer(ITriad[] initial)
    {
        _initial = initial;
    }

    public ITriad[] Optimize()
    {
        //отпимизация методом свертки
        foreach (var triad in _initial)
        {
            if (triad is not SymbolTriad)
            {
                if (triad is BiTriad biTriad)
                {
                    // если оба операнды - константы
                    if (biTriad is { Left: SymbolTriad { Symbol.IsId: false } 
left, Right: SymbolTriad { Symbol.IsId: false } right })
                    {
                        if (biTriad.Operator.Word == ":=")// если 
присваивание, то ищем во всех триадах этот идентификатор и заменяем на 
конечное значение
                        {
                            FindAndChangeId(right.Symbol.Word, left);

                        }
                        else
                        {
                            //считаем константное значение
                            var word = OperateConstants(left, 
biTriad.Operator, right);
                            var sTriad = new SymbolTriad( new 
GrammarSymbol(left.Symbol.Symbol, true, word));
                            FindAndChange(triad, sTriad);
                            _removable.Add(triad); //потом удалим
                        }

                        continue;
                    }
                }

                if (triad is UnoTriad unoTriad)
                {
                    if (unoTriad is { Operand: SymbolTriad symbolTriad, 
Operator.Symbol: "-" })
                    {
                        //подсчет унарного минуса
                        var word = (-
int.Parse(symbolTriad.Symbol.Word)).ToString();
                        var sTriad = new SymbolTriad( new 
GrammarSymbol(symbolTriad.Symbol.Symbol, true, word));
                        FindAndChange(triad, sTriad);
                        _removable.Add(triad);
                        continue;
                    }
                }
```

```
            }
        }

        var list = (new List<ITriad>(_initial));//удаляем лишние триады
        _removable.AddRange(_initial.Where(x=> x is SymbolTriad));
        foreach (var removable in _removable)
        {
            list.Remove(removable);
        }

        return list.ToArray();
    }

    private void FindAndChange(ITriad from, SymbolTriad to)
    {
        foreach (var triad in _initial)
        {
            switch (triad)
            {
                case BiTriad biTriad:
                {
                    if (biTriad.Left == from)
                        biTriad.Left = to;
                    if (biTriad.Right == from)
                        biTriad.Right = to;
                    break;
                }
                case UnoTriad unoTriad:
                {
                    if (unoTriad.Operand == from)
                        unoTriad.Operand = to;
                    break;
                }
            }
        }
    }

    private void FindAndChangeId(string Id, SymbolTriad to)
    {
        foreach (var triad in _initial)
        {
            switch (triad)
            {
                case BiTriad biTriad:
                {
                    if (biTriad.Operator.Word != ":=")
                    {
                        if (biTriad.Left is SymbolTriad lstriad &&
lstriad.Symbol.Word == Id)
                            biTriad.Left = to;
                        if (biTriad.Right is SymbolTriad rstriad &&
rstriad.Symbol.Word == Id)
                            biTriad.Right = to;
                    }
                    else
                    {
                        if (biTriad.Left is SymbolTriad lstriad &&
lstriad.Symbol.Word == Id)
                            biTriad.Left = to;
                    }

                    break;
                }
```

```
                case UnoTriad unoTriad:
                {
                    if (unoTriad.Operand is SymbolTriad striad &&
striad.Symbol.Word == Id)
                        unoTriad.Operand = to;
                    break;
                }
            }
        }
    }

    private string OperateConstants(SymbolTriad left, GrammarSymbol
@operator, SymbolTriad right)
    {
        switch (@operator.Symbol)
        {
            case "{+}":
                return (int.Parse(left.Symbol.Word) +
int.Parse(right.Symbol.Word)).ToString();
            case "{-}":
                return (int.Parse(left.Symbol.Word) -
int.Parse(right.Symbol.Word)).ToString();
            case "{/}":
                return (int.Parse(left.Symbol.Word) /
int.Parse(right.Symbol.Word)).ToString();
            case "{*}":
                return (int.Parse(left.Symbol.Word) *
int.Parse(right.Symbol.Word)).ToString();
        }
        return String.Empty;
    }

}
```

Листинг 4. Генерация Ассемблерного кода по объектному коду

```
public class AssemblerGenerator
{
    public string GenerateCode(ITriad triad)
    {
        //генерация кода по триадам. Берется самая верхняя триада.
        //рекусивным методом генерируется код для каждой триады
        //значение каждой триады после операции записано в ax
        if (triad is BiTriad biTriad)
        {
            switch (biTriad.Operator.Word)
            {
                case "+":
                    return $"{GenerateCode(biTriad.Right)}\n" +
                           "push ax\n" +
                           $"{GenerateCode(biTriad.Left)}\n" +
                           $"pop bx\n" +
                           $"add ax,bx";
                    break;
                case "-":
                    return $"{GenerateCode(biTriad.Right)}\n" +
                           "push ax\n" +
                           $"{GenerateCode(biTriad.Left)}\n" +
                           $"pop bx\n" +
                           $"sub ax,bx";
                    break;
                case "*":
```

```
                    return $"{GenerateCode(biTriad.Right)}\n" +
                           "push ax\n" +
                           $"{GenerateCode(biTriad.Left)}\n" +
                           $"pop bx\n" +
                           $"mul ax,bx";
                    break;
                case "/":
                    return $"{GenerateCode(biTriad.Right)}\n" +
                           "push ax\n" +
                           $"{GenerateCode(biTriad.Left)}\n" +
                           $"pop bx\n" +
                           $"div ax,bx";
                    break;
                case ":=":
                    if(biTriad.Left is SymbolTriad)
                        return $"{GenerateCode(biTriad.Right)}\n" +
                               $"mov {(biTriad.Left as
SymbolTriad).Symbol.Word},ax\n";
                    else
                        return $"{GenerateCode(biTriad.Left)}\n" +
                               $"mov {(biTriad.Right as
SymbolTriad).Symbol.Word},ax\n";
            }
        }

        if (triad is UnoTriad unoTriad)
        {
            switch (unoTriad.Operator.Word)
            {
                case "-":
                    return $"{GenerateCode(unoTriad.Operand)}\n" +
                           $"neg ax\n";
                case ";":
                    return $"{GenerateCode(unoTriad.Operand)}\n";
            }
        }

        if (triad is SymbolTriad symbolTriad)
        {
            return $"mov ax, {symbolTriad.Symbol.Word}";
        }
        return String.Empty;
    }

    public string GenerateCodeForMultyTriad(ITriad[] triads)
    {
        //берем только "верхние" триады, то есть присваивающие
        var toAsm = triads.OfType<BiTriad>().Where(b => b.Operator.Word ==
":=");
        var strbuilder = new StringBuilder();
        foreach (var tr in toAsm)
        {
            strbuilder.Append(GenerateCode(tr));
        }

        return strbuilder.ToString();
    }
}
```