Автоматическая генерация кода для узлов синтаксического дерева и визиторы на изменение синтаксического дерева

Захаренко А.С. (4 курс, 9 группа).
Научный руководитель Михалкович С. С.

Постановка задачи

Целью работы является выполнение следующих подзадач:

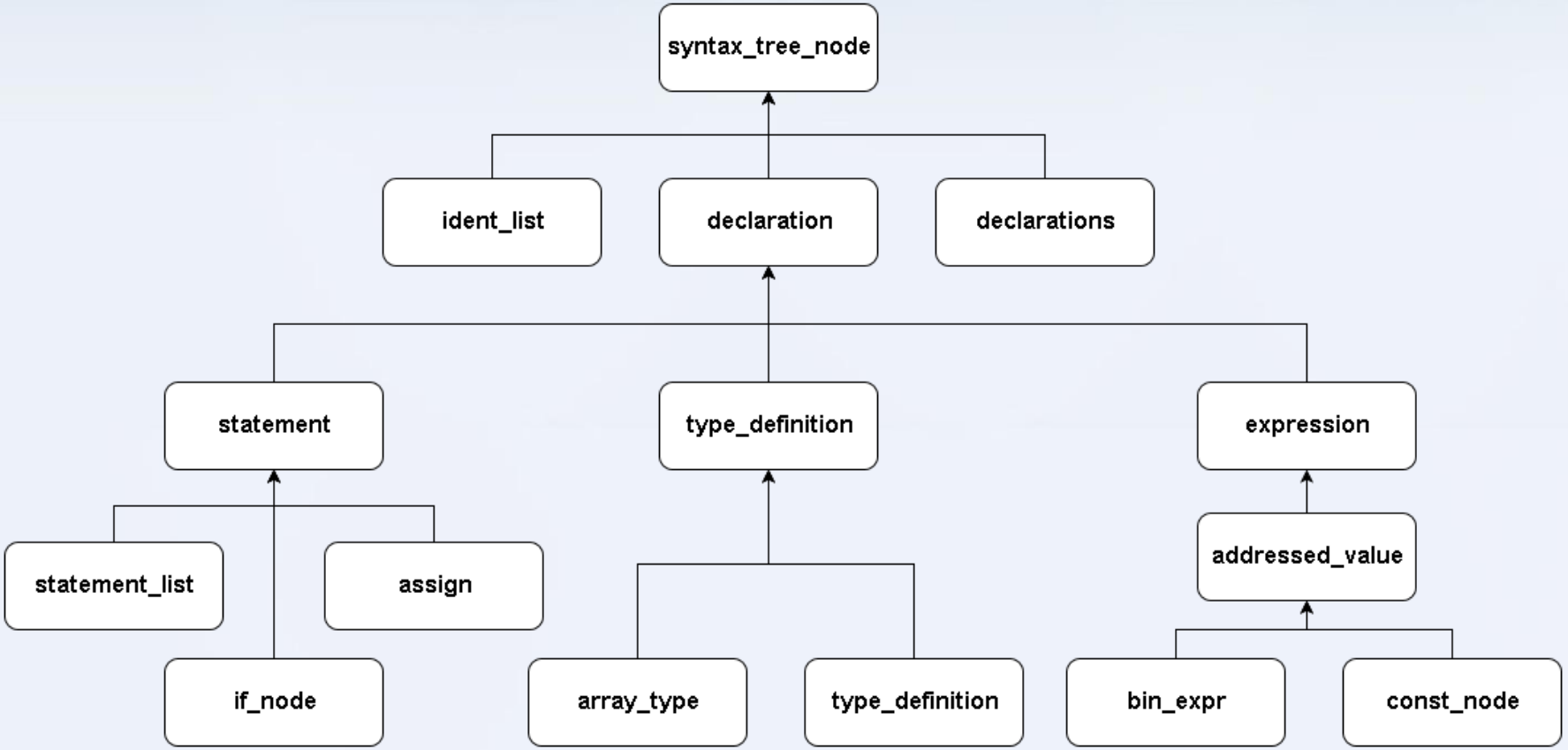
1. Генерация дополнительных методов для синтаксических узлов дерева программы в компиляторе PascalABC.NET, содержащих списки подузлов, по настраиваемому текстовому шаблону

2. Разработка иерархии визиторов, связанных с изменением синтаксического дерева

3. Разработка библиотеки для конструирования наиболее часто встречающихся синтаксических поддеревьев

Иерархия наследования синтаксических узлов

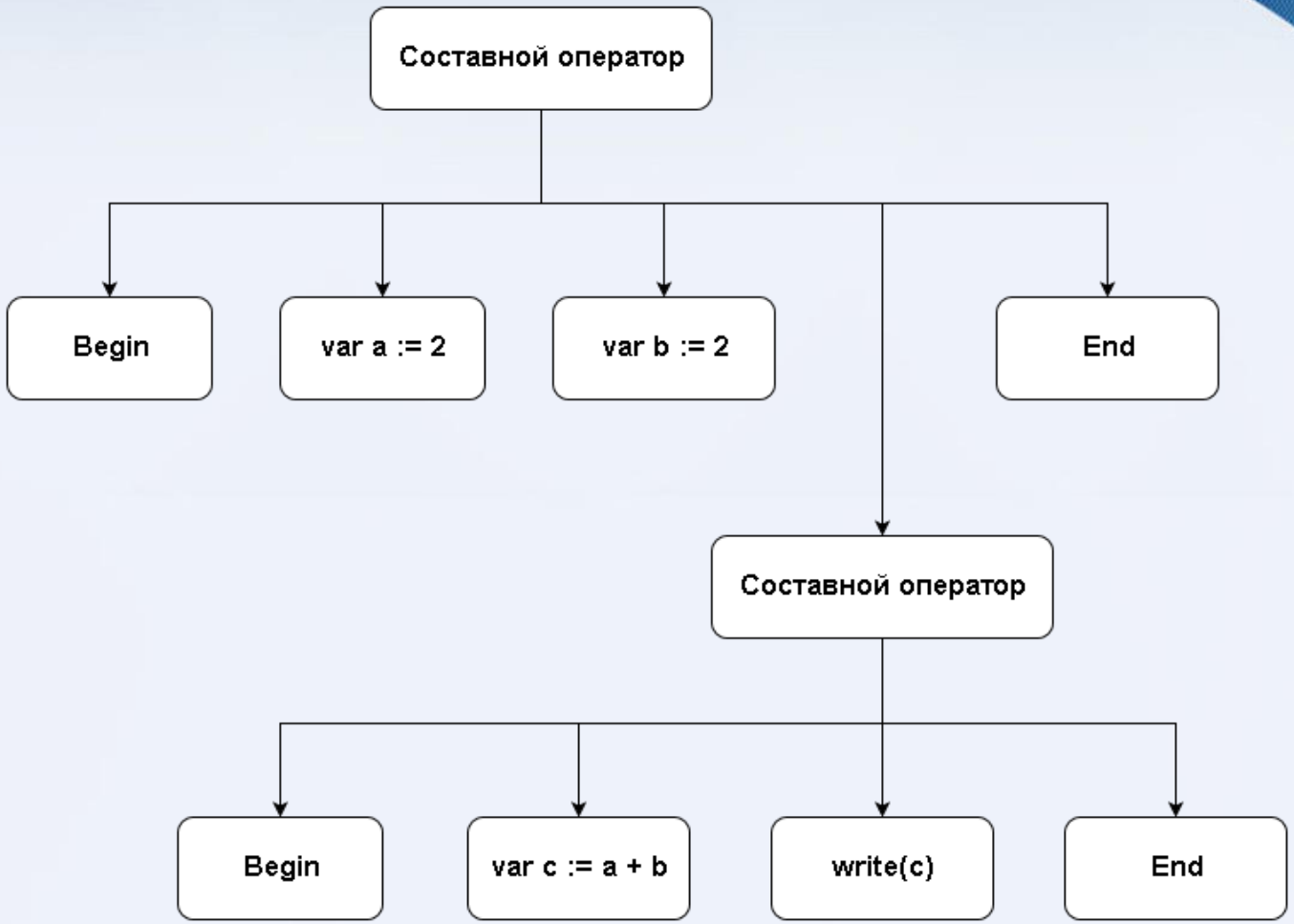
Важнейшие синтаксические узлы:



Удаление лишних begin-end

Пример программы и соответствующего синтаксического дерева:

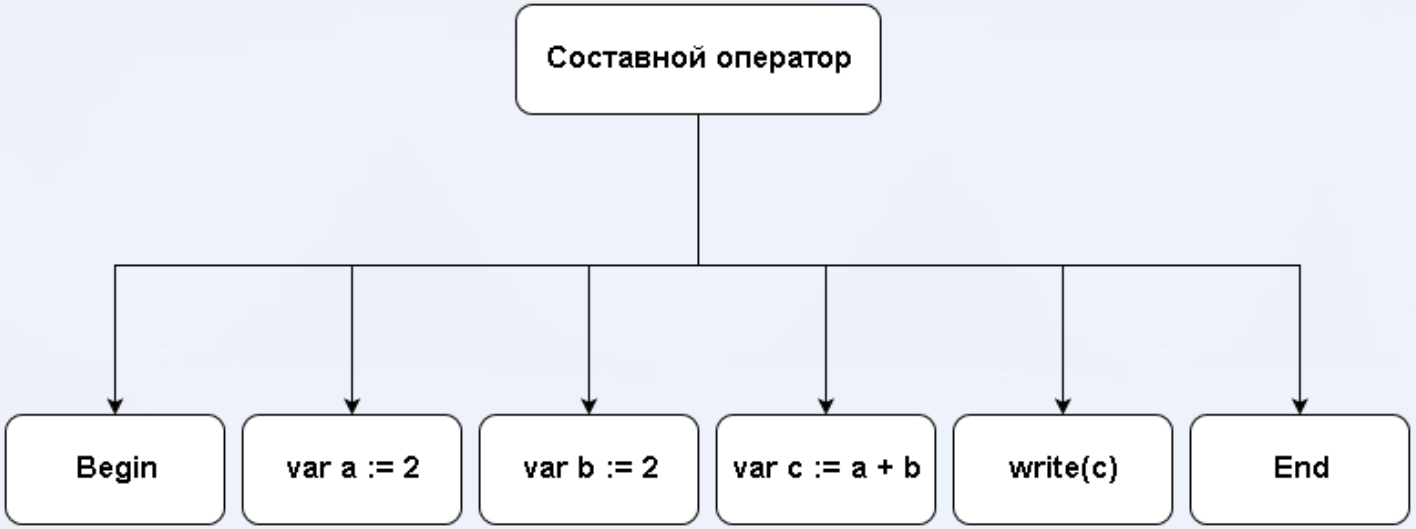
```
begin
  var a := 2;
  var b := 2;
  begin
    var c := a + b;
    write(c);
  end;
end.
```



Удаление лишних begin-end

После прохода по дереву визитора на изменение:

```
begin
  var a := 2;
  var b := 2;
  var c := a + b;
  write(c);
end.
```



Синтаксические узлы со списками

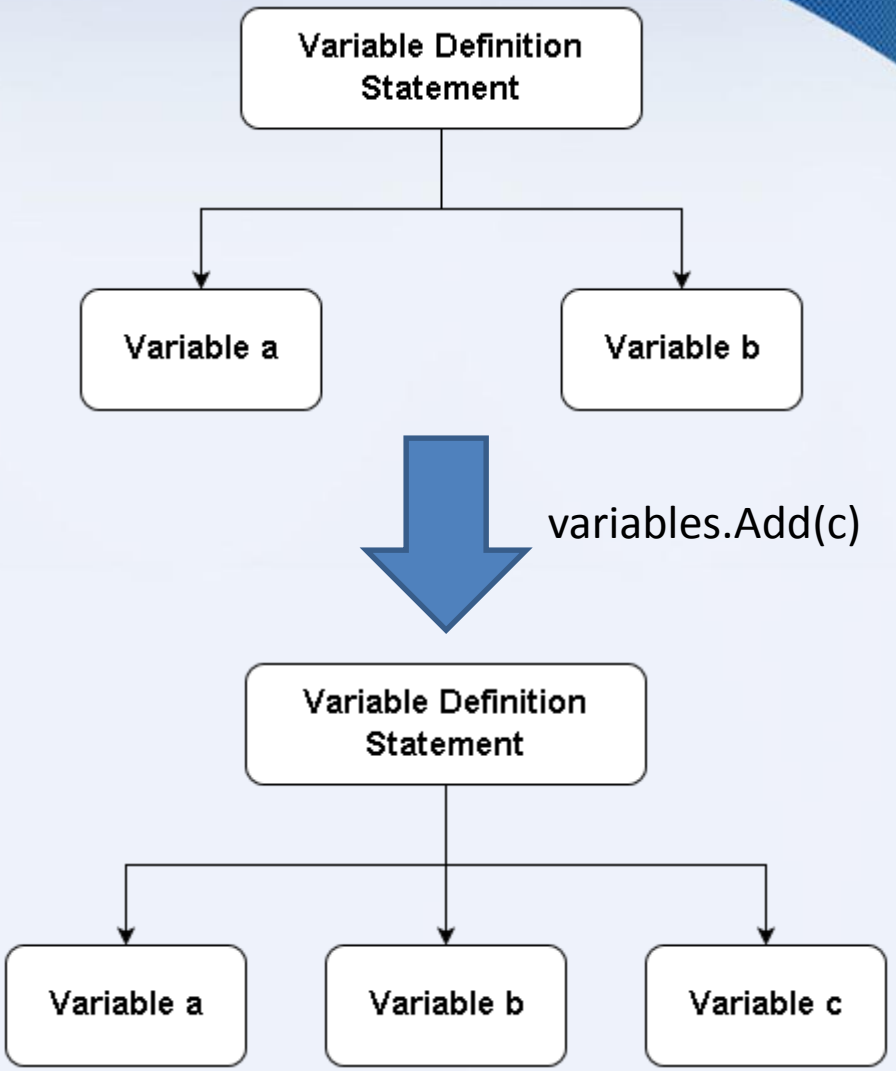
Классы синтаксических узлов, содержащих списки подузлов:

- ident_list
- declarations
- expression_list
- procedure_attributes_list
- consts_definitions_list
- uses_list
- class_members
- class_body
- variant_list
- var_def_list_for_record
- exception_handler_list
- using_list
- named_type_reference_list
- template_param_list
- where_type_specificator_list
- where_definition_list
- enumerator_list
- type_definition_attr_list
- attribute_list
- name_assign_expr_list
- statement_list

Дополнительные методы синтаксических узлов

Функции, сгенерированные для узлов, содержащих списки подузлов :

1. Добавление
2. Удаление
3. Поиск
4. Замена



Генерация по шаблону

«list_name» и «list_element_type» – переменные шаблона

Пример генерации для класса «список идентификаторов»

```
public void AddFirst(list_element_type el)
{
     list_name.Insert(0, el);
}

private int FindIndex(list_element_type el)
{
     var ind = list_name.FindIndex(x => x == el);
     if (ind == -1)
         throw new Exception(""Элемент не найден"");
     return ind;
}

public bool Remove(list_element_type el)
{
     return list_name.Remove(el);
}
```

```
public void AddFirst(ident el)
{
    idents.Insert(0, el);
}

private int FindIndex(ident el)
{
    var ind = idents.FindIndex(x => x == el);
    if (ind == -1)
        throw new Exception("Элемент не найден");
    return ind;
}

public bool Remove(ident el)
{
    return idents.Remove(el);
}
```

Иерархия базовых визиторов

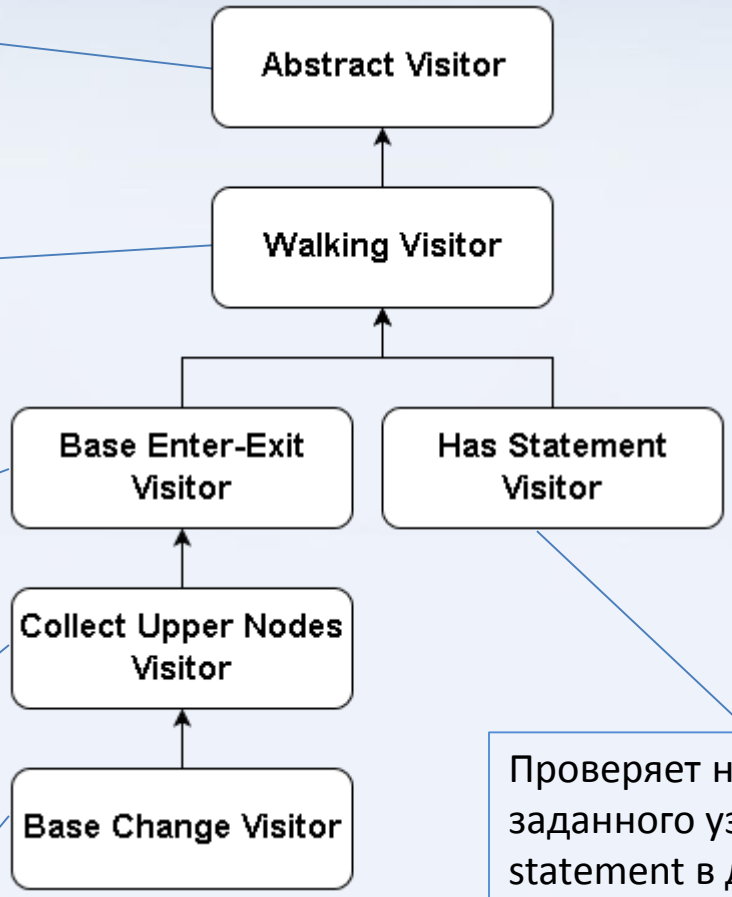
Для каждого узла описаны свои действия, выполняемые при посещении

Содержит делегаты для выполнения действий над узлом при входе и при выходе

Объявляет делегаты переопределяемыми в наследниках методами

Накапливает информацию о предках при обходе узлов

Предоставляет базовые возможности редактирования узлов



Проверяет наличие заданного узла типа statement в дереве

Прикладные визиторы

Название визитора	Назначение
Pretty Printer Visitor	Восстанавливает исходный текст программы с учетом правил форматирования
Count Nodes Visitor	Подсчитывает количество узлов каждого типа в дереве
Delete Local Defs	Удаляет объявления локальных переменных
Delete Redundant Begin-Ends	Удаляет излишние объявления составных операторов
Lowering Visitor	Заменяет циклы на условные операторы и безусловные переходы

Abstract Visitor

Описание и возможности:

- Обходит все узлы дерева
- Для каждого узла описаны свои действия, выполняемые при посещении
- Позволяет определить действие по умолчанию при обходе узла

```
public virtual void DefaultVisit(syntax_tree_node n)
{
}

public virtual void visit(statement _statement)
{
        DefaultVisit(_statement);
}
```

Walking Visitor

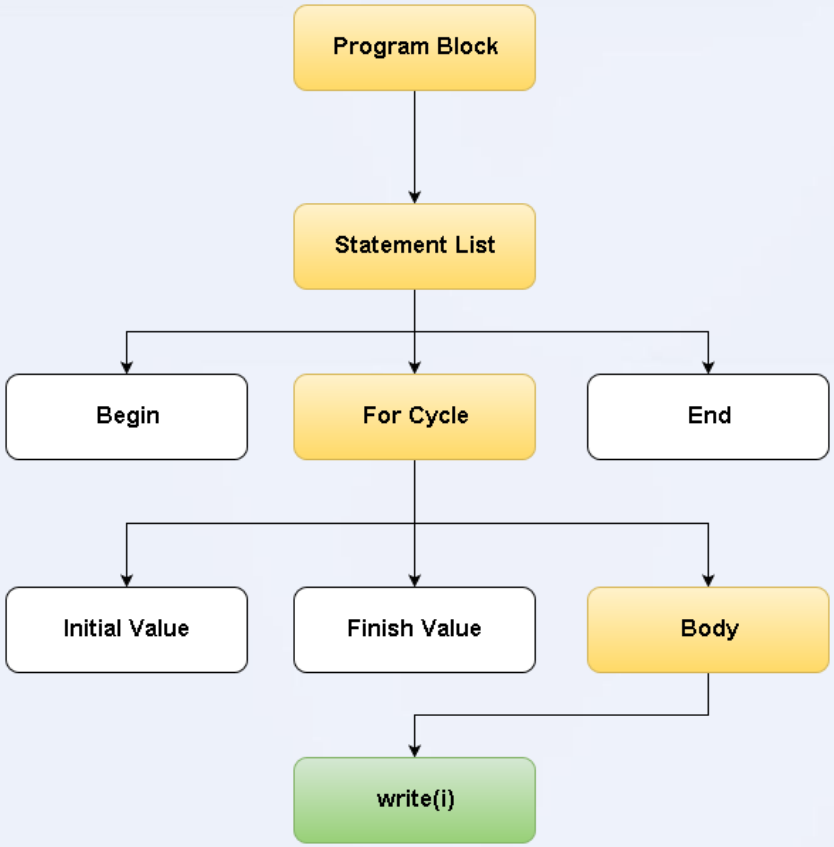
Описание и возможности:

- Содержит переопределяемые действия, совершаемые перед и после посещения узла
- Позволяет посещать не все узлы (флаг visitNode контроллирует обход узла)

```
public virtual void ProcessNode(syntax_tree_node Node)
{
        if (Node != null)
        {
                if (OnEnter != null)
                      OnEnter(Node);

                if (visitNode)
                        Node.visit(this);
                else visitNode = true;

                if (OnLeave != null)
                        OnLeave(Node);
        }
}

public override void DefaultVisit(syntax_tree_node n)
{
        var count = n.subnodes_count;
        for (var i = 0; i < count; i++)
                ProcessNode(n[i]);
}
```

Collect Upper Nodes Visitor

```
begin
  for var i := 0 to 5 do
    write(i); <- текущая позиция визитора
end.
```



Визитор накапливает все узлы по пути от текущей позиции до корня.

В данном случае накоплены следующие узлы:

Base Change Visitor

Часть методов базового визитора на изменение:

```
public void Replace(syntax_tree_node from, syntax_tree_node to)
{
        var upper = UpperNode(); // обращение к предку
        if (upper == null)
                throw new Exception("У корневого элемента нельзя получить UpperNode");
        upper.Replace(from, to); // использование методов, встроенных в узел
}

public T UpperNodeAs<T>(int up = 1) where T : syntax_tree_node
{
        var stl = UpperNode(up) as T;
        if (stl == null)
                throw new Exception("Элемент вложен не в " + typeof(T));
        return stl;
}

public void ReplaceStatement(statement from, IEnumerable<statement> to)
{
        var stl = UpperNodeAs<statement_list>();
        stl.Replace(from, to);
}
```

Lowering-визитор

- Замена циклов на условные операторы и метки «goto»
- Переопределение поведения для необходимых узлов
- Наличие фильтра узлов

Переопределение действия, совершаемого перед началом обхода узла. В условном операторе указываются узлы, которые нужно обойти. Для остальных флаг visitNode принимает значение false и они не обходятся визитором:

```
public override void Enter(syntax_tree_node st)
{
        base.Enter(st);
        if (!(st is procedure_definition || st is block || st is statement_list || st is case_node ||
              st is for_node || st is foreach_stmt || st is if_node || st is repeat_node ||
             st is while_node || st is with_statement || st is try_stmt || st is lock_stmt))
               {
                       visitNode = false;
               }
}
```

Lowering-визитор

Пример реализации обхода узла while_node с заменой цикла на if и безусловные переходы:

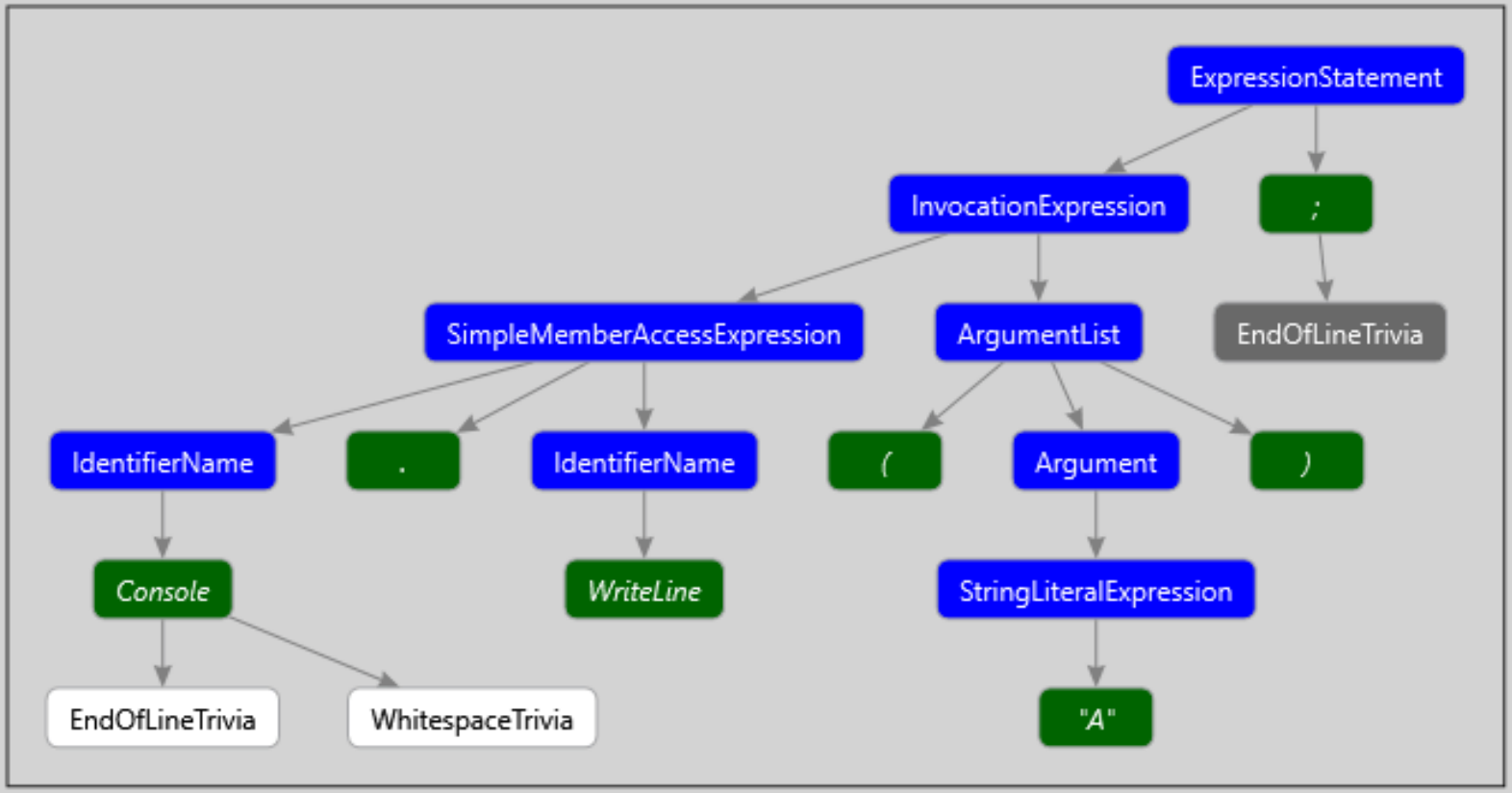
```
public override void visit(while_node wn)
{
        ProcessNode(wn.statements);
        var b = HasStatementVisitor<yield_node>.Has(wn);
        if (!b)
                return;

        var gt1 = goto_statement.New;
        var gt2 = goto_statement.New;

        var if0 = new if_node(un_expr.Not(wn.expr), gt1);
        var lb2 = new labeled_statement(gt2.label, if0);
        var lb1 = new labeled_statement(gt1.label);

        ReplaceStatement(wn, SeqStatements(lb2, wn.statements, gt2, lb1));

        // в declarations ближайшего блока добавить описание labels
        block bl = listNodes.FindLast(x => x is block) as block;

        bl.defs.Add(new label_definitions(gt1.label, gt2.label));
}
```

Задача построения синтаксических поддеревьев

- Исследование возможностей построения поддеревьев в Roslyn, свободном компиляторе C# компании Microsoft.
- Создание построителя синтаксических поддеревьев для языка компилятора PascalABC.NET

Фабрика синтаксических узлов Roslyn

Синтаксическое дерево вызова функции «Console.WriteLine("A");»



Фабрика синтаксических узлов Roslyn

Создание синтаксического дерева, представляющего код «Console.WriteLine("A");»

```
return SyntaxFactory.ExpressionStatement(
           SyntaxFactory.InvocationExpression(
               SyntaxFactory.MemberAccessExpression(
                   SyntaxKind.SimpleMemberAccessExpression,
                   SyntaxFactory.IdentifierName(
                       @"Console"),
                   SyntaxFactory.IdentifierName(
                       @"WriteLine"))
               .WithOperatorToken(
                   SyntaxFactory.Token(
                       SyntaxKind.DotToken)))
           .WithArgumentList(
               SyntaxFactory.ArgumentList(
                   SyntaxFactory.SingletonSeparatedList<ArgumentSyntax>(
                       SyntaxFactory.Argument(
                           SyntaxFactory.LiteralExpression(
                               SyntaxKind.StringLiteralExpression,
                               SyntaxFactory.Literal(
                                   SyntaxFactory.TriviaList(),
                                   @"""A""",
                                   @"""A""",
                                   SyntaxFactory.TriviaList()))))))
               .WithOpenParenToken(
                   SyntaxFactory.Token(
                       SyntaxKind.OpenParenToken))
               .WithCloseParenToken(
                   SyntaxFactory.Token(
                       SyntaxKind.CloseParenToken))));
```

Строитель синтаксических поддеревьев

Возможности строителя:

- Создание вызова процедуры/функции
- Объявление процедуры/функции
- Создание циклов
- Создание простых классов
- Объявление переменных

Создание синтаксического дерева, представляющего код «write('A');»

```
CreateProcedureCall("write", new char_const('A'));
```

Результаты работы

В процессе разработки были решены все три поставленные задачи:

1. Была реализована генерация дополнительных методов для синтаксических узлов дерева программы в компиляторе PascalABC.NET, содержащих списки подузлов, по настраиваемому текстовому шаблону

2. Реализована иерархия визиторов, позволяющих изменять синтаксическое дерево в процессе обхода. Визиторы на изменение обращаются к методам узлов.

3. Разработана библиотека построения часто используемых синтаксических поддеревьев.

Ссылка на проект в системе github

https://github.com/