МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

«Ульяновский государственный технический университет**»**

Кафедра «Вычислительная техника**»**

**Лабораторная работа №3**

по дисциплине «Системное программное обеспеченье»

«Семантический анализ»

Выполнил студент

гр. ИВТАСбд-31

Сутулов К. О.

Ульяновск, 2022

# Задание

На основе предыдущих работ необходимо реализовать семантический анализатор для языка программирования “C”. Семантический анализатор должен проверять правильность использования поддерживаемых типов данных.

# Ход работы

Синтаксический анализатор из предыдущей работы поддерживает числовые (char, int, float), логические (bool) и строковые (string) типы данных.

На основании поддерживаемых типов данных были определены следующие правила типизации:

1) Каждому идентификатору может быть присвоено значение соответствующего типа.

2) К любому числовому типу данных может быть приведено значение любого другого числового типа данных.

3) К логическому типу может быть приведено любое значение числового типа

4) Арифметические операции (+, -, \*, /) доступны только с числовыми данными, и их результат приводится к максимальному по размеру числовому типу, операция (+) также может применяться при сложении двух строковых типов.

5) Результат логических операций (!, ==, !=, <=, =>, &&, ||, <, >) применимы к логическим и числовым типам, их результат всегда логического типа. Операции (==, !=) также применимы к строковым типам данных.

6) Любой другой вариант выбрасывает исключение.

Для реализации семантического анализа был модифицирован класс-родитель всех синтаксических узлов BaseNode – в него был добавлен метод определния числового типа IsNumeric, метод, возвращающий максимальный по размеру числовой тип, а также перегружаемый метод Maximize, реализация которого должна возвращать тип выражения:

public class BaseNode

{

public BaseNode()

{

}

private List<string> \_numericTypes = new() { "bool", "char", "int", "float" };

protected string MaxNumType(string type1, string type2)

{

if (!IsNumeric(type1) || !IsNumeric(type2)) throw new Exception("Type error: expected numeric type");

int i1 = \_numericTypes.IndexOf(type1);

int i2 = \_numericTypes.IndexOf(type2);

return (i1 > i2) ? type1 : type2;

}

protected bool IsNumeric(string type)

{

return \_numericTypes.Contains(type);

}

public virtual string Maximize()

{

throw new Exception("Not implemented");

}

}

Затем в каждом из узлов, содержащим результат выражения, перегружаем метод Maximize. Для узлов констант и идентификаторов сразу можно вернуть тип содержимого значения. Также можно выполнять проверки типов сразу в конструкторе узлов.

public class IdentifierNode : ExpressionNode

{

public string Type;

public IdentifierNode() { }

public IdentifierNode(string id, string type) : base(id)

{

Type = type;

}

public override string Maximize()

{

return Type;

}

}

Для узла унарных операций (-, !) в методе Maximize проведем проверку на строковый тип, с которым их нельзя использовать. Реузльтат операции (!) – логический тип, а унарного минуса - числовой:

public class UnaryNode : ExpressionNode

{

public ExpressionNode Expression;

public UnaryNode() { }

public UnaryNode(string op, ExpressionNode expression) : base(op)

{

Expression = expression;

Maximize();

}

public override string Maximize()

{

string type = Expression.Maximize();

if ((Operation == "-" || Operation == "!") && type == "string" )

{

throw new Exception($"Cannot use operation {Operation} with type string");

}

if (Operation == "!")

{

return "bool";

}

return type;

}

}

Для узла инкремента в конструкторе выполняем проверку на числовой тип:

public class IncrementNode : ExpressionNode

{

public string Identifier;

public string Type;

public IncrementNode() { }

public IncrementNode(string op, string id, string type) : base(op)

{

Identifier = id;

Type = type;

if (!IsNumeric(type))

{

throw new Exception("Type error: cannot increment non-numeric value");

}

}

}

Для узлов с условиями в конструкторе выполняем проверку, что тип выражения можно привести к логическому (логический и числовой типы):

public IfNode(ExpressionNode condition, StatementNode then, StatementNode? els = null)

{

this.Condition = condition;

this.Then = then;

this.Else = els;

string type = Condition.Maximize();

if (!IsNumeric(type))

{

throw new Exception($"Cannot convert type {type} to bool");

}

}

Для узла бинарных операций задаем правила применения в соответствии с требованиями приведения типов:

public class BinaryNode : ExpressionNode

{

public ExpressionNode Expression1, Expression2;

public string Type;

public BinaryNode() { }

public BinaryNode(string op, ExpressionNode expression1, ExpressionNode expression2) : base(op)

{

Expression1 = expression1;

Expression2 = expression2;

Type = Maximize();

}

public override string Maximize()

{

string type1 = Expression1.Maximize();

string type2 = Expression2.Maximize();

string op = base.Operation;

if ((op == "==" || op == "!=") && type1 == "string" && type2 == "string")

{

return "bool";

}

if ((op == "&&" || op == "||" || op == "==" || op == ">" || op == "<" || op == ">=" || op == "<=") && type1 != "string" && type2 != "string")

{

return "bool";

}

if ((op == "+" || op == "-" || op == "+" || op == "-"))

{

if (type1 == "string" || type2 == "string")

{

if (type1 != type2 || op != "+") throw new Exception("Type error: strings can only be added");

return "string";

}

return MaxNumType(type1, type2);

}

throw new Exception($"Type error: cannot make operation \"{op}\" with types \"{type1}\", \"{type2}\"");

}

}

Наконец задаются правила приведения типов в узле присвоения. В методе Check проверяется, может ли тип выражения быть приведен к типу идентификатора:

namespace lab3.Models.SyntaxNodes

{

public class SetNode : StatementNode

{

public string Name;

public ExpressionNode Expression;

public string Type;

public SetNode() { }

public SetNode(string name, ExpressionNode expression, string type)

{

Name = name;

Expression = expression;

Type = type;

Check();

}

public void Check()

{

string expressionType = Expression.Maximize();

if (

(Type == "bool" && expressionType != "string") ||

(Type == "string" && expressionType == "string") ||

(IsNumeric(expressionType) && IsNumeric(Type)) ||

(IsNumeric(Type) && expressionType == "bool")

)

{

return;

}

throw new Exception($"Type error: cannot cast type \"{expressionType}\" into \"{Type}\"");

}

}

}

# Тестирование программы

Сначала семантический анализатор проверялся на корректном примере. Следующие программы успешно прошли проверку типов:

Пример 1

{

int a = 10;

string b = "test";

if (b == "test" && a > 5) {

a++;

}

}

Пример 2

{

int a = 10 + true + '5';

string b = "Hello" + " " + "Wolrd";

do {

a = a + 1;

b = b + "!";

} while(a > 10);

}

Затем проверялись ошибочные примеры. Далее представлены не прошедшие проверку типов программы, а также вывод транслятора:

Пример 3

{

string a = "User" + 123;

}

**System.Exception:** "Type error: strings can only be added with each other"

Пример 4

{

string b = 5;

b++;

}

**System.Exception:** "Type error: cannot cast type "int" into "string""

Пример 5

{

string b = "true";

b++;

}

**System.Exception:** "Type error: cannot increment non-numeric value"

Пример 6

{

int a = 5;

string b = "true";

do {

a++;

} while (b);

}

**System.Exception:** "Cannot convert type string to bool"

# Вывод

В процессе выполнения лабораторной работы с использованием платформы .NET была составлена программная реализация семантического анализа (проверки типов) языка “С”.