Федеральное государственное бюджетное образовательное

учреждение высшего образования

«Алтайский государственный технический университет

им. И.И. Ползунова»

Факультет (институт) Информационных технологий

Кафедра Прикладная математика

Отчет защищен с оценкой\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Е.Н.Крючкова

(подпись преподавателя) (инициалы, фамилия)

“\_\_\_\_”\_\_\_\_\_\_\_\_\_\_\_ 2020 г.

Расчётное задание

по дисциплине Методы анализа в теории формальных языков

(наименование дисциплины)

ЛР 09.03.04.07.000 ОТ

(обозначение документа)

Студент группы ПИ-61 К.А.Дуплинская

(инициалы, фамилия)

Преподаватель профессор Е.Н.Крючкова

(должность, ученое звание) (инициалы, фамилия)

Барнаул 2020

1. **Задание**

Программа: главная программа языка С++. Допускается описание структур.

Типы данных: int, float.

Операции: арифметические и сравнения.

Операторы: присваивания и for.

Операнды: простые переменные, элементы структур и константы.

Константы: целые в 10 c/c, вещественные с фиксированной точкой.

1. **Определение синтаксически управляемого перевода.**

СУ-перевод основывается на том, что смысл предложения языка зависит от синтаксической структуры этого предложения. Она справедлива как для формальных языков, так и естественных. Главное отличие заключается в том, что естественные языки допускают неоднозначности в грамматиках, а в языках программирования неоднозначности недопустимы. Любое правило ЯП имеет четкую структуру и однозначный смысл, напрямую связанный с этой структурой.

СУ-перевод – это основной метод порождения кода результирующей программы на основании результатов синтаксического анализа. Для удобства понимания сути метода можно считать, что результат синтаксического анализа представлен в виде дерева синтаксического анализа, хотя в реальных компиляторах это не всегда так.

Целью СУ-перевода является совместная запись синтаксических и семантический правил языка в такой форме, которая пригодна для реализации интерпретации или трансляции в процессе грамматического разбора.

В данной работе нам необходимо правильно встроить множество семантический функций в правила языка, тем самым построить перевод.

1. **Ограничения и замечания.**

**Ограничения и замечания нисходящего анализа.**

В нисходящем анализе нельзя использовать грамматику с левой рекурсией, поэтому необходимо избавиться от нее перед СУ-переводом.

**Ограничения и замечания восходящего анализа.**

Основное ограничение восходящего анализа – это невозможность встраивания Δ-функций в середину правила, поскольку мы не можем выполнить семантическую функцию, пока не знаем какое правило мы отсканировали. Поэтому семантические функции необходимо навешивать на терминальные символы, если они находятся не в конце правила языка.

**Общие замечания.**

Для вычисления типов должен использоваться магазин типов.

Приведение типов отличается в нисходящем анализе отличается от приведения в восходящем потому, что операнды бинарных операций находятся в разных правилах.

1. **Построение СУ-перевода.**

**Исходная грамматика для нисходящего перевода.**

<P> → <A><P> | ε

<A> → <D> | <S> **|** <F>

<S> → struct TIdent {<E>};

<D> → <J> <I> ;

<E> → <D><E> | ε

<J> → int | float | TIdent

<I> → <G> <I1>

<I1> → , <I> | ε

<G> → TIdent <K>

<K> → = V | ε

<F> → <J> Tident () <M>

<M> → { <Op> }

<O> → ; | <M> | <Q> | <W>; | <VF> | return <V>;

<Q> → for (<W>; <V>; <V>) <O>

<W> → <N> <W1>

<W1> → = <V>

<Op> → <D><Op> | <S><Op> | <O><Op> | ε

<V> → <Sr><Z>

<Z> → == <Sr><Z>| != <Sr><Z> | ε

<Sr> → <Sum><Z1>

<Z1> → **<** <Sum><Z1> | **>** <Sum><Z1> | **<=** <Sum><Z1> | **>=** <Sum><Z1> | ε

<Sum> → <Mul><Z2>

<Z2> → + <Mul><Z2> | – <Mul><Z2> | ε

<Mul> → <SV><Z3>

<Z3>→ \* <SV><Z3> | / <SV><Z3> | % <SV><Z3> | ε

<SV> → <N> | <C> | <VF> | (V)

<VF> → TIdent ()

<N> → TIdent <N1>

<N1> → . TIdent | ε

<C> → TConstInt | TConstFloat

**СУ-перевод для нисходящего анализа.**

<P> → <A><P> | ε

<A> → <D> | <S> **|** <F>

<S> → struct TIdent ***ΔsetStruct*** {<E>}; ***ΔreturnLevel***

<D> → <J> ***ΔtypeVar*** <I>; ***ΔendDecl***

<E> → <D><E> | ε

<J> → int ***ΔstartDecl*** | float ***ΔstartDecl*** | TIdent ***ΔstartDecl***

<I> → <G><I1>

<I1> → , <I> | ε

<G> → TIdent ***ΔsetVar*** <K>

<K> → = <V> ***ΔAssign*** | ε

<F> → <J> ***ΔtypeVar*** Tident ***ΔsetFunc*** () <M> ***ΔreturnLevel***

<M> → { ***ΔsetNewLevel*** <Op> } ***ΔreturnLevel***

<O> → ; | <M> | <Q> | <W>; | <VF> | return <V>; ***ΔcheckTypeReturn***

<Q> → for ***ΔsetNewLevel*** (<W>; <V>; <V>) <O> ***ΔreturnLevel***

<W> → <N><W1>

<W1> → = <V> ***ΔAssign***

<Op> → <D><Op> | <S><Op> | <O><Op> | ε

<V> → <Sr><Z>

<Z> → == <Sr> ***Δoper*** <Z>| != <Sr> ***Δoper*** <Z> | ε

<Sr> → <Sum><Z1>

<Z1> → **<** <Sum> ***Δoper*** <Z1> | **>** <Sum> ***Δoper*** <Z1> | **<=** <Sum> ***Δoper*** <Z1> | **>=** <Sum> ***Δoper*** <Z1> | ε

<Sum> → <Mul><Z2>

<Z2> → + <Mul> ***Δoper*** <Z2> | – <Mul> ***Δoper*** <Z2> | ε

<Mul> → <SV><Z3>

<Z3> → \* <SV> ***Δoper*** <Z3> | / <SV> ***Δoper*** <Z3> | % <SV> ***Δoper*** <Z3> | ε

<SV> → <N> | <C> | <VF> | (V)

<VF> → TIdent ***ΔfindFunc*** ()

<N> → TIdent ***ΔfindVar*** <N1>

<N1> → . <N> | ε

<C> → TConstInt ***ΔconstType*** ***Δpush\_t*** | TConstFloat ***ΔconstType*** ***Δpush\_t***

ΔstartDecl – установить флаг описания данных и запомнить тип в глобальной переменной dataType.

ΔendDecl – сбросить флаг описания данных.

ΔsetVar – Sem4(a, t) – проверяет на дублирование и заносит идентификатор *а* в качестве переменной с типом данных *t*.

ΔsetFunc – Sem2(a, t) – проверяет на дублирование и заносит идентификатор а в качестве имени функции и возвращает тип *t*.

ΔsetNewLevel – начало нового уровня.

ΔreturnLevel – Sem3(k) – восстанавливает текущее значения указателя таблицы по значению *k*.

ΔcheckTypeReturn – Sem7(t) – проверка типа возвращаемого значения.

ΔconstType – определение типа константы.

Δpush\_t – положить тип в магазин типов.

Δoper – Sem9(t, g) – вычисляет тип результат операции над операндами *t* и *g* в соответствии с таблицей приведений.

ΔfindVar – Sem6.2(a, t) – нахождение объекта *a* в таблице идентификаторов, определение типа этого объекта и возврат этого типа в качестве результата *t*. При отсутствии переменной *a* в таблице выдается сообщение о семантической ошибке.

ΔfindFunc – Sem6.1(a, t) – нахождение объекта *a* в таблице идентификаторов, определение типа этого объекта и возврат этого типа в качестве результата *t*. При отсутствии идентификатора a в таблице выдается сообщение о семантической ошибке. Если не функция - ошибка

ΔAssign – Sem8(t, g) – проверка на соответствие топов переменной (t) и выражения (g)

ΔtypeVar – Sem1(lex, t) – по изображению типа определяет и возвращает семантический тип

ΔsetStruct – Sem5(a) – проверяет на дублирование и заносит *а* в качестве имени структуры

**Исходная грамматика для восходящего перевода.**

<P> → <P><A> | ε

<A> → <D> | <S> | < F>

<S> → struct TIdent {<E>};

<D> → <J><I>;

<E> → <E> <D> | ε

<J> → int | float | TIdent

<I> → <I>, <G> | <G>

<G> → TIdent | <K>

<K> → TIdent = <V>

< F> → <J> Tident () <M>

<M>→ { <Op>}

<O> → ; | <M> | <Q> | <W>; | <VF>

<Q> → for (<W> ; <V> ; <V>) <O>

<W> → <N> = <V>

<Op>→ <Op><D> | <Op><S> | <Op> <O> | ε

<V> → <V> == <Sr> | <V> != <Sr>| <Sr>

<Sr>→ <Sr> **<** <Sum> | <Sr> **>** <Sum> | <Sr> **<=** <Sum> | <Sr> **>=** <Sum> | <Sum>

<Sum> → <Sum> + <Mul> | <Sum> – <Mul> | <Mul>

<Mul> → <Mul> \* <SV>| <Mul> / <SV>| <Mul> % <SV>| <SV>

<SV> → <N> | <C> | <VF> | (<V>)

<VF> → TIdent ()

<N> → <N> . TIdent | TIdent

<C>→ TConstInt | TConstFloat

**СУ-перевод для восходящего анализа**

<P> → <P><A> | ε

<A> → <D> | <S> | < F>

<S> → struct TIdent {<E>}; ***ΔreturnLevel f(TIdent) = ΔsetStruct***

<D> → <J><I>; ***ΔendDecl*** ***f(int) = ΔtypeVar,***

***f(float) = ΔtypeVar***

<E> → <E> <D> | ε

<J> → int | float | TIdent ***f(int) = f(float)***

***=f(TIdent) = ΔstartDecl***

<I> → <I>, <G> | <G>

<G> → TIdent | <K> ***f(TIdent) = ΔsetVar***

<K> → TIdent = <V>***ΔAssign***

< F> → <J> Tident () <M> ***ΔreturnLevel*** ***f(int) = ΔtypeVar,***

***f(float) = ΔtypeVar***

***f(TIdent) = ΔsetFunc***

<M>→ { <Op>} ***ΔreturnLevel*** ***f({)=ΔsetNewLevel***

<O> → ; | <M> | <Q> | <W>; | <VF> | return <V>; ***ΔcheckTypeReturn***

<Q> → for (<W> ; <V> ; <V>) <O> ***ΔreturnLevel***  ***f(‘for’) = ΔsetNewLevel***

<W> → <N> = <V> ***ΔAssign(t(N), t(V))***

<Op>→ <Op><D> | <Op><S> | <Op> <O> | ε

<V> → <V> == <Sr> ***f(V) = Δoper(t(V), t(Sr), sign)***

| <V> != <Sr> ***f(V) = Δoper(t(V), t(Sr), sign)***

| <Sr> ***f(V) = t(Sr),***

<Sr>→ <Sr> **<** <Sum> ***f(Z) = Δoper(t(Sr1), t(Sum), sign)***

| <Sr> **>** <Sum> ***f(Z) = Δoper(t(Sr1), t(Sum), sign)***

| <Sr> **<=** <Sum> ***f(Z) = Δoper(t(Sr1), t(Sum), sign)***

| <Sr> **>=** <Sum> ***f(Z) = Δoper(t(Sr1), t(Sum), sign)***

| <Sum> ***f(Z) = t(Sum)***

<Sum> → <Sum> + <Mul> ***f(Z1) = Δoper(t(Sum1), t(Mul), sign)***

| <Sum> – <Mul> ***f(Z1) = Δoper(t(Sum1), t(Mul), sign)***

| <Mul> ***f(Z1) = t(Mul)***

<Mul> → <Mul> \* <SV> ***f(Z2) = Δoper(t(Mul1), t(SV), sign)***

| <Mul> / <SV> ***f(Z2) = Δoper(t(Mul1), t(SV), sign)***

| <Mul> % <SV> ***f(Z2) = Δoper(t(Mul1), t(SV), sign)***

| <SV> ***f(Z2) = Δoper(t(Mul1), t(SV), sign)***

<SV> → <N> | <C> | <VF> | (<V>)

<VF> → TIdent () ***f(TIdent) = ΔfindFunc(TIdent)***

<N> → <N> . TIdent | TIdent ***f(TIdent) = ΔfindVar(TIdent)***

<C>→ TConstInt ***ΔconstType*** | TConstFloat ***ΔconstTyp***

ΔstartDecl – установить флаг описания данных и запомнить тип в глобальной переменной dataType.

ΔendDecl – сбросить флаг описания данных.

ΔsetVar – Sem4(a, t) – проверяет на дублирование и заносит идентификатор *а* в качестве переменной с типом данных *t*.

ΔsetFunc – Sem2(a, t) – проверяет на дублирование и заносит идентификатор а в качестве имени функции и возвращает тип *t*.

ΔsetNewLevel – начало нового уровня.

ΔreturnLevel – Sem3(k) – восстанавливает текущее значения указателя таблицы по значению *k*.

ΔcheckTypeReturn – Sem7(t) – проверка типа возвращаемого значения.

ΔconstType – определение типа константы.

Δpush\_t – положить тип в магазин типов.

Δoper – Sem9(t, g) – вычисляет тип результат операции над операндами *t* и *g* в соответствии с таблицей приведений.

ΔfindVar – Sem6.2(a, t) – нахождение объекта *a* в таблице идентификаторов, определение типа этого объекта и возврат этого типа в качестве результата *t*. При отсутствии переменной *a* в таблице выдается сообщение о семантической ошибке.

ΔfindFunc – Sem6.1(a, t) – нахождение объекта *a* в таблице идентификаторов, определение типа этого объекта и возврат этого типа в качестве результата *t*. При отсутствии идентификатора a в таблице выдается сообщение о семантической ошибке. Если не функция - ошибка

ΔAssign – Sem8(t, g) – проверка на соответствие топов переменной (t) и выражения (g)

ΔtypeVar – Sem1(lex, t) – по изображению типа определяет и возвращает семантический тип

ΔsetStruct – Sem5(a) – проверяет на дублирование и заносит *а* в качестве имени структуры