Министерство науки и высшего образования РФ

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.

Отчет

по расчётному заданию

по дисциплине

Теория языков программирования и методы трансляции

(наименование дисциплины)

ЛР 09.03.04.14.000 ОТ

(обозначение документа)

Студент группы ПИ-61 В.Р. Оверченко

(инициалы, фамилия)

Преподаватель профессор, доцент Е.Н. Крючкова

(должность, ученое звание) (инициалы, фамилия)

Барнаул 2020

Оглавление

[2. Контекстные условия транслируемого языка 4](#_Toc38104838)

[2.1. Типы данных 4](#_Toc38104839)

2.2. Контекстные условия………………………………………………………………4

2.3. Операционные символы……………………………………………………………5

[3. СУ-перевод для нисходящего синтаксического анализа 7](#_Toc38104840)

[3.1. Ограничения СУ-перевода для нисходящего синтаксического анализа 7](#_Toc38104841)

[3.2. Нисходящий СУ-перевод 7](#_Toc38104842)

[4. СУ-перевод для восходящего синтаксического анализа 9](#_Toc38104843)

[4.1. Ограничения СУ-перевода для восходящего синтаксического анализа 9](#_Toc38104844)

[4.2. Восходящий СУ-перевод 9](#_Toc38104845)

1. **Задание**

Программа: главная программа языка С++. Допускается описание функций без параметров, функции возвращают значения допустимых в программе типов.

Типы данных: double, char.

Операции: арифметические и сравнения.

Операторы: присваивания и while.

Операнды: простые переменные, элементы одномерных массивов и константы.

Константы: строковые, символьные и целые в 10 c/c .

# Контекстные условия транслируемого языка

# Типы данных

|  |  |  |
| --- | --- | --- |
| Тип | Разрядность | Диапазон значений |
| double | 64 разряда | от +/- 1.7E-308 до +/- 1.7E+308 |
| char | 8 разрядов | От -128 до 127 |

**Таблица приведений:**

|  |  |  |  |
| --- | --- | --- | --- |
| Первый тип | Второй тип | Приведение типов | Результат |
| double | double | Нет приведения | double |
| char | double | char приводится к double и вычисляется | double |
| char | char | Нет приведения | double |
| double | char | char приводится к double вычисляется | double |

Целые константы в выражении сразу приводятся к double.

* 1. **Контекстные условия**

Типы объектов и поля необходимые для их хранения:

• Функции (идентификатор, тип возвращаемого значения, указатель на начало функции);

• Простые переменные (идентификатор, тип данных и флаг инициализации);

• Элементы массивов (идентификатор, тип данных и флаг инициализации, индекс);

• Пустой тип.

Таким образом получаем общую структуру полей для всех типов объектов:

* Идентификатор;
* Тип данных;
* Указатель на начало функции;
* Флаг инициализации;
* Индекс массива.
  1. **Операционные символы**

**∆type** **=** Sem0(lex) **-** определение семантического типа объекта по изображение лексемы lex.

**∆identArr =** Sem2(lex, t) – добавление идентификатора массива lex с типом t в таблицу, если в таблице уже есть идентификатор lex на данном уровне, то печатаем сообщение об ошибке.

**∆identVar** = Sem1(lex, t) – добавление идентификатора переменной lex с типом t в таблицу, если в таблице уже есть идентификатор lex на данном уровне, то печатаем сообщение об ошибке.

**∆identFunc** = Sem5(lex, t) – добавление идентификатора функции lex с типом возвращаемого результата t в таблицу, если в таблице уже есть идентификатор lex на данном уровне, то печатаем сообщение об ошибке.

**∆find =** Sem11\_Find (lex) – поиск объекта в таблице идентификаторов, определение типа объекта и возвращение этого тип в качестве результата t. При отсутствии lex в таблице идентификаторов печатаем сообщение об ошибке и возвращаем «неопределённый тип».

**∆assign =** Sem12\_Assign (t, d) – проверяет соответствие типа выражения d с типом идентификатора t.

**∆res\_type =** Sem9\_Res\_type (op1, op2) **-** вычисление типа результата выполнения арифметических и логических действий над операндами op1, op2 в соответствии с таблицей приведения.

**∆unar =** Sem8\_Unar (t) **-** проверка на допустимость выполнения унарной арифметической операции + или – над типом данных t.

**∆inc\_dec =** Sem7\_Inc\_dec (t) **-** проверка на допустимость выполнения инкрементирования и декрементирования над данными (тип double или целая константа, иначе ошибка) типа t.

**∆getTypeConst =** Sem10\_Type\_const (t) – определение и возвращение типа константы.

**∆block\_beg=** Sem13\_Block\_beg() - создание 2 пустых вершин для блока данных.

**∆block\_end=** Sem14\_Block\_end() - достать из верхушки стека блок данных, и вернуть с него указатель в семантической таблице.

**∆pop** – очищение верхушки магазина.

**∆index** = Sem15\_Index (i) – проверяет , что выражение, используемое в качестве индекса массива, является целочисленной константой.

**∆checkIndex** = Sem16\_check\_Index (i) – проверка на совпадение индекса i с номером индекса, указанного при объявлении.

# СУ-перевод для нисходящего синтаксического анализа

# Ограничения СУ-перевода для нисходящего синтаксического анализа

Нисходящий синтаксический анализатор должен выполнять левый разбор, соответствующий левому выводу. Для этого над грамматикой необходимо выполнить следующие действия:

1. Избавиться от левой рекурсии, заменив правила

A → Az1|Az2|...|Azn|y1|y2|...|ym на совокупность правил

A → y1D|y2D|...|ymD; D → z1D|z2D|...|znD|ε.

1. Общие множители в правилах необходимо вынести. Простейший алгоритм устранения общих левых множителей основан на замене правил A → yx1|yx2|...|yxm|z1|z2|...|zk на правила A → yD|z1|z2|...|zk; D → x1|x2|...|xm.
2. Скрытые левые множители также следует вынести. Выявить скрытые левые множители сложнее, чем обнаружить нарушения типа (1) и (2). Однако при некотором опыте анализа КС–грамматик это легко сделать. Например, правила A → Bx|y; B → yz можно заменить на правила A → yT; T → z|ε; B → yz.

При выполнении всех трёх правил, можно использовать грамматику для СУ-перевода по стратегии нисходящего синтаксического разбора.

# Нисходящий СУ-перевод

S --> TS | ε

T --> W | F

W --> double ∆type D; | char ∆type D;

D --> Z D1

D1 --> , Z D1 | ε

Z --> id ∆identVar Z1 | R Z1

Z1 --> = A ∆Assign | ∆pop ε

R --> id ∆identArr [c1 ∆index]

F --> double ∆type Y () Q ∆block\_end | char ∆type Y () Q ∆block\_end

Y --> main ∆identFunc | id ∆identFunc

Q --> {∆block\_beg O}∆block\_end

O --> W O | K O | ε

K --> U; | Q | P | H | ;

P --> id ∆find ();

H --> while (A) K

U --> id ∆find = A ∆Assign | M | N

M --> ++ G ∆inc\_dec; | -- G ∆inc\_dec; | G M1

M1 --> ++∆inc\_dec; | -- ∆inc\_dec;

N --> G N1

N1 --> += A ∆Assign | -= A ∆Assign | \*= A ∆Assign | /= A ∆Assign | %= A ∆Assign

G --> id ∆find J

J --> [A ∆index] ∆checkIndex | ε

A --> B A1 | + B ∆unar A1 | - B ∆unar A1 |

A1 --> > B ∆Assign A1 | >= B ∆Assign A1 | < B ∆Assign A1 | <= B ∆Assign A1 | == B ∆Assign A1 | A != B ∆Assign A1 | ε

B --> + B1 ∆ res\_type | - B1 ∆ res\_type | E

B1 --> + E ∆res\_type |- E ∆res\_type

E --> X E1

E1 --> \* X ∆res\_type E1 | / X ∆res\_type E1 | % X ∆res\_type E1 | ε

X --> G | C | (A) |

C --> c1 ∆getTypeConst | c2 ∆getTypeConst | c3 ∆getTypeConst

# СУ-перевод для восходящего синтаксического анализа

# Ограничения СУ-перевода для восходящего синтаксического анализа

Операционные символы можно применять только в конце правила.

Если есть необходимость применить операционные символы в середине правила, то нужно использовать функцию перевода от терминальных символов, после которых необходимо вставить операционный символ.

# Восходящий СУ-перевод

S --> ST | ε

T -->W | F

W -->double D; f(double) = ∆type | char D; f(char) = ∆type

D --> D, Z | Z

Z --> id ∆pop | id = A ∆assign | R = A ∆assign | R ∆pop f(id) = ∆identVar

R --> id [c1] f(id) = ∆identArr f1(]) = ∆index

F --> double Y () Q ∆block\_end f(double) = ∆type | char Y () Q ∆block\_end f(char) = ∆type

Y --> main | id f(id) = f(main) = ∆identFunc

Q --> {O} ∆block\_end f( { ) = ∆block\_beg

O --> O W | O K | ε

K --> U; | Q | P | H | ;

P --> id (); f(id) = ∆find

H --> H (A) K

U --> id = A; f(id) = ∆find | M | N

M --> G ++; ∆inc\_dec (t(G), sign) | ++ G; ∆inc\_dec (t(G), sign) | G --; ∆inc\_dec (t(G), sign) | -- G; ∆inc\_dec (t(G), sign)

N --> G += A; ∆assign (t(G), t(A), sign) | G -= A; ∆assign (t(G), t(A), sign) | G \*= A; ∆assign (t(G), t(A), sign) | G /= A; ∆assign (t(G), t(A), sign) | G %= A; ∆assign (t(G), t(A), sign)

G --> J | id f(id) = ∆find

J --> id [A] ∆checkIndex f(id) = ∆find f1(]) = ∆index

A --> A > B ∆assign (t(A), t(B), sign) | A >= B ∆assign (t(A), t(B), sign) | A < B ∆assign (t(A), t(B), sign) | A <= B ∆assign (t(A), t(B), sign) | A == B ∆assign (t(A), t(B), sign) | A != B ∆assign (t(A), t(B), sign) | + B t(A) = ∆unar (t(B), sign) | - B t(A) = ∆unar (t(B), sign) | B t(A) = t(B)

B --> B + E t(B1) = ∆res\_type (t(B2), t(E), sign) | B - E t(B1) = ∆res\_type (t(B2), t(E), sign) | E t(B1) = t(E)

E --> E \* X t(E1) = ∆res\_type(t(E2), t(X), sign) | E / X t(E1) = ∆res\_type(t(E2), t(X), sign) | E % X t(E1) = ∆res\_type(t(E2), t(X), sign) | X t(E1) = t(X)

X --> G | C | (A) |

C --> c1∆getTypeConst | c2 ∆getTypeConst | c3 ∆getTypeConst