Минобрнауки России

ФГБОУ ВПО «НИУ МЭИ» АВТИ

Кафедра математического и компьютерного моделирования

**Курсовая работа**

**Математическое обеспечение ЭВМ**

**«Построение транслятора»**

**Работу выполнил:**

Солонин Егор А-14-19

Вариант 14

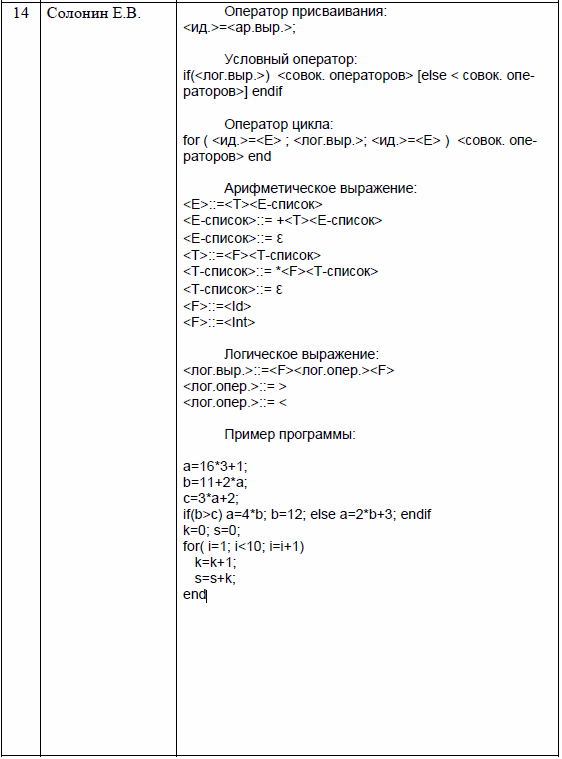
**Работу принял:**

Князев А. В.

Москва 2021

**Задание:**

1. Преобразовать заданную грамматику в L-атрибутную транслирующую грамматику в форме простого присваивания.
2. Разработать объектный язык.
3. Разработать атрибутный автомат для трансляции предложений данной грамматики.
4. Разработать программу, реализующую МП-транслятор. Программа разрабатывается как приложение с графическим интерфейсом на языке C# в среде Visual Studio.
5. Транслятор разрабатывается как программа, реализующая алгоритмы, изложенные в теоретическом курсе. Не должны использоваться вспомогательные средства: коллекции, регулярные выражения, функции разбора строк и т.д.
6. Разработать программу-интерпретатор для созданного объектного языка

*Индивидуальный вариант (№14):*

**LL(1)-грамматика:**

1. <программа>::= <оператор><спис. операторов>
2. <спис. операторов>::= <оператор><спис. операторов>
3. <спис. операторов>::=
4. <оператор>::= <идент.>=<E>;
5. <оператор>::=for(<идент.>=<E>; <лог.выр.>; <идент.>=<E>) <спис. операторов> end
6. <оператор>::= if(<лог.выр.>) <спис. операторов> <иначе> endif
7. <иначе>::= else <спис. операторов>
8. <иначе>::=
9. <лог.выр.>::= <F><лог.опер.><F>
10. <лог.опер.>::= >
11. <лог.опер.>::= <
12. <E>::= <T><E-список>
13. <E-список>::= +<T><E-список>
14. <E-список>::=
15. <T>::= <F><T-список>
16. <T-список>::=  \*<F><T-список>
17. <T-список>::=
18. <F>::= <идент.>
19. <F>::= <целое>

**Транслирующая грамматика:**

1. <программа> -> <оператор><спис. операторов>
2. <спис. операторов> -> <оператор><спис. операторов>
3. <спис. операторов> ->
4. <оператор> -> <идент.>=<E>{Присвоить};
5. <оператор> ->

for(<идент.>=<E>{Присвоить};

{Метка}<лог.выр.> {Усл. переход по 0}{Безусловн. переход};

{Метка}<идент.>=<E>{Присвоить}{Безусловн. переход};)

{Метка} <спис. операторов> {Безусловн. переход} end {Метка}

1. <оператор> -> if(<лог.выр.>){Усл. переход по 0} <спис. операторов> <иначе> endif
2. <иначе> -> else {Метка}{Безусловн. переход} <спис. операторов> {Метка}
3. <иначе> -> {Метка}
4. <лог.выр.> -> <F><лог.опер.>
5. <лог.опер.> -> > <F> {Больше}
6. <лог.опер.> -> < <F> {Меньше}
7. <E> -> <T><E-список>
8. <E-список> -> +<T>{Сложить}<E-список>
9. <E-список> ->
10. <T> -> <F><T-список>
11. <T-список> ->  \*<F>{Умножить}<T-список>
12. <T-список> ->
13. <F> -> <идент.>{Id}
14. <F> -> <целое>{Int}

**Атрибутная транслирующая грамматика:**

1. <программа> -> <оператор><спис. операторов>
2. <спис. операторов> -> <оператор><спис. операторов>
3. <спис. операторов> ->
4. <оператор> -> <идент.>p1=<E>q1{Присвоитьp2,q2};

p2 := p1, q2 := q1

1. <оператор> ->

for(<идент.>p1=<E>q1{Присвоитьp2,q2};

{Меткаv2}<лог.выр.>y1 {Усл. переход по 0y2,w1}{Безусловн. Переходx1};

{Меткаz2}<идент.>p3=<E>q3{Присвоитьp4,q4}{Безусловн. Переходv1};)

{Меткаx2} <спис. операторов> {Безусловн. Переход­z1} end {Метка­w2}

(p2, p3, p4) := p1

(q2, q3, q4) := q1

y2 := y1

(z1, z2) := НовМ

(w1, w2) := НовМ

(x1, x2) := НовМ

(v1, v2) := НовМ

1. <оператор> -> if(<лог.выр.>p1){Усл. переход по 0p2,z1} <спис. операторов> <иначе>z2 endif

p2 := p1, (z2, z1) := НовМ

1. <иначе>z1 -> else {Безусловн. Переходw1}{Меткаz2} <спис. операторов> {Меткаw2}

z2 := z1, (w2, w1) := НовМ

1. <иначе>z1 -> {Меткаz2}

z2 := z1

1. <лог.выр.>t2 -> <F>p1<лог.опер.>p2,t1

p2 := p1 t2 := t1

1. <лог.опер.>p1,t2 -> > <F>q1 {Большеp2,q2,r1}

p2 := p1 q2 := q1 (r1, t2) := Нов

1. <лог.опер.>p1,t2 -> < <F>q1 {Меньшеp2,q2,r1}

p2 := p1 q2 := q1 (r1, t2) := Нов

1. <E>t2 -> <T>p1<E-список>p2,t1

p2 := p1 t2 := t1

1. <E-список>p1,t2 -> +<T>q1{Сложитьp2,q2,r1}<E-список>r2,t1

p2 := p1 q2 := q1 (r2, r1) := Нов t2 := t1

1. <E-список>p1,t2 ->

t2 := p1

1. <T>t2 -> <F>p1<T-список>p2,t1

p2 := p1 t2 := t1

1. <T-список>p1,t2 ->  \*<F>q1{Умножитьp2,q2,r1}<T-список>r2,t1

p2 := p1 q2 := q1 (r2, r1) := Нов t2 := t1

1. <T-список>p1,t2 ->

t2 := p1

1. <F>q2 -> <идент.>q1

q2 := q1

1. <F>q2 -> <целое>q1

q2 := q1

**Формализованное построение множеств выбора:**

*Аннулирующие нетерминалы и правила*

<спис. операторов>, <E список>, <T список>, <else>

*Начинается-прямо-с*

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | <пр-ма> | <опер.> | <ар. выр.> | <Е> | <Есп> | <Т> | <Тсп> | <F> | <лог. выр.> | <идент.> | <конст.> | for | if | else | = | ; | \* | ( | ) | + | < | > |
| <пр-ма> |  | 1 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| <опер.> |  |  |  |  |  |  |  |  |  | 1 |  | 1 | 1 |  |  |  |  |  |  |  |  |  |
| <ИНАЧЕ> |  |  |  |  |  |  |  |  |  |  |  |  |  | 1 |  |  |  |  |  |  |  |  |
| <Е> |  |  |  |  |  | 1 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| <Есп> |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | 1 |  |  |
| <Т> |  |  |  |  |  |  |  | 1 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| <Тсп> |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | 1 |  |  |  |  |  |
| <F> |  |  |  |  |  |  |  |  |  | 1 | 1 |  |  |  |  |  |  |  |  |  |  |  |
| <лог. выр.> |  |  |  |  |  |  |  | 1 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| <лог. оп.> |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | 1 | 1 |

*Начинается-с*

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | <пр-ма> | <опер.> | <Е> | <Есп> | <Т> | <Тсп> | <F> | <лог. Выр> | <лог. оп.> | <идент.> | <конст.> | for | if | else | = | ; | \* | ( | ) | + | < | > |
| <пр-ма> | 1 | 1 |  |  |  |  |  |  |  | 1 |  | 1 | 1 |  |  |  |  |  |  |  |  |  |
| <опер.> |  | 1 |  |  |  |  |  |  |  | 1 |  | 1 | 1 |  |  |  |  |  |  |  |  |  |
| <Е> |  |  | 1 |  | 1 |  | 1 |  |  | 1 | 1 |  |  |  |  |  |  |  |  |  |  |  |
| <Есп> |  |  |  | 1 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | 1 |  |  |
| <Т> |  |  |  |  | 1 |  | 1 |  |  | 1 | 1 |  |  |  |  |  |  |  |  |  |  |  |
| <Тсп> |  |  |  |  |  | 1 |  |  |  |  |  |  |  |  |  |  | 1 |  |  |  |  |  |
| <F> |  |  |  |  |  |  | 1 |  |  | 1 | 1 |  |  |  |  |  |  |  |  |  |  |  |
| <лог. выр.> |  |  |  |  |  |  | 1 | 1 |  | 1 | 1 |  |  |  |  |  |  |  |  |  |  |  |
| <лог. оп.> |  |  |  |  |  |  |  |  | 1 |  |  |  |  |  |  |  |  |  |  |  | 1 | 1 |

*Прямо-перед*

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | <пр-ма> | <опер.> | <Е> | <Есп> | <Т> | <Тсп> | <F> | <лог. Выр> | <лог. оп.> | <идент.> | <конст.> | for | if | else | = | ; | \* | ( | ) | + | < | > |
| <пр-ма> |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| <опер.> | 1 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| <Е> |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | 1 |  |  |  |  |  |  |
| <Есп> |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| <Т> |  |  |  | 1 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| <Тсп> |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| <F> |  |  |  |  |  | 1 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| <лог. выр.> |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | 1 |  |  | 1 |  |  |  |
| <лог. оп.> |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| <идент.> |  |  |  |  |  |  |  |  |  |  |  |  |  |  | 1 |  |  |  |  |  |  |  |
| <конст.> |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| if |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | 1 |  |  |  |  |
| else |  | 1 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| = |  |  | 1 |  |  |  | 1 | 1 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| ; |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| \* |  |  |  |  |  |  | 1 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| ( |  |  |  |  |  |  |  | 1 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| ) |  | 1 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| + |  |  |  |  | 1 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| < |  |  |  |  |  |  |  |  | 1 |  |  |  |  |  |  |  |  |  |  |  |  |  |
| > |  |  |  |  |  |  |  |  | 1 |  |  |  |  |  |  |  |  |  |  |  |  |  |

*Прямо-на-конце*

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | <пр-ма> | <опер.> | <ар. выр.> | <Е> | <Есп> | <Т> | <Тсп> | <F> | <лог. Выр> | <лог. оп.> | <идент.> | <конст.> | for | if | else | = | ; | \* | ( | ) | + | < | > |
| <пр-ма> |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| <опер.> | 1 | 1 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| <ар. выр.> |  | 1 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| <Е> |  |  | 1 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| <Есп> |  |  |  | 1 | 1 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| <Т> |  |  |  | 1 | 1 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| <Тсп> |  |  |  |  |  | 1 | 1 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| <F> |  |  |  |  |  | 1 | 1 |  |  | 1 |  |  |  |  |  |  |  |  |  |  |  |  |  |
| <лог. выр.> |  |  |  |  |  |  |  |  | 1 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| <лог. оп.> |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| <идент.> |  |  |  |  |  |  |  | 1 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| <конст.> |  |  |  |  |  |  |  | 1 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| if |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| else |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| = |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| ; |  | 1 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

*На-концe*

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | <пр-ма> | <опер.> | <ар. выр.> | <Е> | <Есп> | <Т> | <Тсп> | <F> | <лог. Выр> | <лог. оп.> | <идент.> | <конст.> | for | if | else | = | ; | \* | ( | ) | + | < | > |
| <пр-ма> | 1 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| <опер.> |  | 1 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| <ар. выр.> |  | 1 | 1 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| <Е> |  | 1 | 1 | 1 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| <Есп> |  | 1 | 1 | 1 | 1 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| <Т> |  |  | 1 | 1 | 1 | 1 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| <Тсп> |  |  | 1 | 1 | 1 | 1 | 1 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| <F> |  |  | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |  |  |  |  |  |  |  |  |  |  |  |  |  |
| <лог. выр.> |  |  |  |  |  |  |  |  | 1 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| <лог. оп.> |  |  |  |  |  |  |  |  |  | 1 |  |  |  |  |  |  |  |  |  |  |  |  |  |
| <идент.> |  |  | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |  |  |  |  |  |  |  |  |  |  |  |  |
| <конст.> |  |  | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |  | 1 |  |  |  |  |  |  |  |  |  |  |  |
| = |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | 1 |  |  |  |  |  |  |  |
| ; | 1 | 1 |  |  |  |  |  |  |  |  |  |  |  |  |  |  | 1 |  |  |  |  |  |  |

*Перед*

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | <пр-ма> | <опер.> | <ар. выр.> | <Е> | <Есп> | <Т> | <Тсп> | <F> | <лог. Выр> | <лог. оп.> | <идент.> | <конст.> | for | if | else | = | ; | \* | ( | ) | + | < | > |
| <пр-ма> |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| <опер.> |  | 1 |  |  |  |  |  |  |  |  | 1 |  | 1 | 1 |  |  |  |  |  |  |  |  |  |
| <ар. выр.> |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | 1 |  |  |  |  |  |  |
| <Е> |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | 1 |  |  |  |  |  |  |
| <Есп> |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | 1 |  |  |  |  |  |  |
| <Т> |  |  |  |  |  | 1 |  |  |  |  | 1 | 1 |  |  |  |  | 1 |  |  |  |  |  |  |
| <Тсп> |  |  |  |  | 1 |  |  |  |  |  |  |  |  |  |  |  | 1 |  |  |  |  |  |  |
| <F> |  |  |  |  | 1 |  | 1 |  |  |  |  |  |  |  |  |  | 1 | 1 |  |  |  |  |  |
| <лог. выр.> |  |  |  |  |  |  |  |  | 1 |  | 1 | 1 |  |  |  |  | 1 |  |  |  |  |  |  |
| <лог. оп.> |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | 1 |  | 1 |  |  | 1 | 1 |
| <идент.> |  |  |  |  | 1 |  | 1 | 1 |  | 1 |  |  |  |  |  | 1 | 1 | 1 | 1 |  |  | 1 | 1 |
| <конст.> |  |  |  |  | 1 |  | 1 | 1 |  | 1 |  |  |  |  |  | 1 | 1 | 1 | 1 |  |  | 1 | 1 |
| for |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| if |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | 1 |  |  |  |  |
| else |  | 1 |  |  |  |  |  |  |  |  | 1 |  | 1 | 1 | 1 |  |  |  |  |  |  |  |  |
| = |  |  | 1 | 1 |  | 1 | 1 |  |  |  | 1 | 1 |  |  |  |  |  | 1 |  |  |  |  |  |
| ; |  | 1 |  |  |  |  |  |  |  |  | 1 |  | 1 | 1 |  |  |  |  |  |  |  |  |  |
| \* |  |  |  |  |  |  | 1 |  |  |  |  |  |  |  |  |  |  | 1 |  |  |  |  |  |
| ( |  |  |  |  |  |  |  |  |  |  | 1 | 1 |  |  |  |  |  |  |  |  |  |  |  |
| ) |  | 1 |  |  |  |  |  |  |  |  | 1 |  | 1 | 1 |  |  |  |  |  |  |  |  |  |
| + |  |  |  |  |  | 1 |  |  |  |  | 1 | 1 |  |  |  |  |  |  |  |  |  |  |  |
| < |  |  |  |  |  |  |  |  |  |  | 1 | 1 |  |  |  |  |  |  |  |  |  |  |  |
| > |  |  |  |  |  |  |  |  |  |  | 1 | 1 |  |  |  |  |  |  |  |  |  |  |  |

**Множества выбора:**

Выбор (1) = Перв(<оператор><спис. операторов>) = {<идент.>, if, for}

Выбор (2) = Перв(<оператор><спис. операторов>) = {<идент.>, if, for}

Выбор (3) = След(<спис. операторов>) = {else, endif, end, “-|”}

Выбор (4) = Перв(<идент.>=<E>;) = {<идент.>}

Выбор (5) = Перв(for(<идент.>=<E>;<лог.выр.>;<идент.>=<E>)<спис. операторов>end) = {for}

Выбор (6) = Перв(if(<лог.выр.>) <спис. операторов> <иначе> endif) = {if}

Выбор (7) = Перв(else <спис. операторов>) = {else}

Выбор (8) = След(<иначе>) = {<идент.>, for, if, endif, end, “-|”}

Выбор (9) = Перв(<F><лог.опер.><F>) = {<идент.>, <целое>}

Выбор (10) = Перв(>) = {>}

Выбор (11) = Перв(<) = {<}

Выбор (12) = Перв(<T><E-список>) = {<идент.>, <целое>}

Выбор (13) = Перв(+<T><E-список>) = {+}

Выбор (14) = След(<E-список>) = {;}

Выбор (15) = Перв(<F><T-список>) = {<идент.>, <целое>}

Выбор (16) = Перв(\*<F><T-список>) = {\*}

Выбор (17) = След(<T-список>) = {+, ;}

Выбор (18) = Перв(<идент.>) = {<идент.>}

Выбор (19) = Перв(<целое>) = {<целое>}

**Управляющая таблица МП-автомата:**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Магаз. символы | <идент> | <целое> | for | if | else | endif | end | + | \* | ( | ) | > | < | = | ; | -| |
| <программа> | 1 | O | 1 | 1 | O | O | O | O | O | O | O | O | O | O | O | O |
| <спис. операторов> | 2 | O | 2 | 2 | 3 | 3 | 3 | O | O | O | O | O | O | O | O | 3 |
| <оператор> | 4 | O | 5 | 6 | O | O | O | O | O | O | O | O | O | O | O | O |
| <else> | 8 | O | 8 | 8 | 7 | 8 | 8 | O | O | O | O | O | O | O | O | 8 |
| <лог.выр.> | 9 | 9 | O | O | O | O | O | O | O | O | O | O | O | O | O | O |
| <лог. опер.> | O | O | O | O | O | O | O | O | O | O | O | 10 | 11 | O | O | O |
| <E> | 12 | 12 | O | O | O | O | O | O | O | O | O | O | O | O | O | O |
| <E список> | O | O | O | O | O | O | O | 13 | O | O | O | O | O | O | 14 | O |
| <T> | 15 | 15 | O | O | O | O | O | O | O | O | O | O | O | O | O | O |
| <T список> | O | O | O | O | O | O | O | 16 | 16 | O | O | O | O | O | 17 | O |
| <F> | 18 | 19 | O | O | O | O | O | O | O | O | O | O | O | O | O | O |
| <идент.> | # | O | O | O | O | O | O | O | O | O | O | O | O | O | O | O |
| <endif> | O | O | O | O | O | # | O | O | O | O | O | O | O | O | O | O |
| <end> | O | O | O | O | O | O | # | O | O | O | O | O | O | O | O | O |
| = | O | O | O | O | O | O | O | O | O | O | O | O | O | # | O | O |
| ( | O | O | O | O | O | O | O | O | O | # | O | O | O | O | O | O |
| ) | O | O | O | O | O | O | O | O | O | O | # | O | O | O | O | O |
| ; | O | O | O | O | O | O | O | O | O | O | O | O | O | O | # | O |
| if | O | O | O | O | # | O | O | O | O | O | O | O | O | O | O | O |
| for | O | O | O | O | O | O | O | O | O | O | O | O | O | O | O | O |
| ˅ | О | О | О | О | О | O | O | О | О | О | О | О | О | O | O | Д |

Начальное содержимое магазина: <программа>˅

1. Заменить(<оператор><спис. операторов>), Держать
2. Заменить(<оператор><спис. операторов>), Держать
3. Вытолкнуть, Держать
4. Заменить(=<E>;), Сдвиг
5. Заменить(for( <идент.>=<E>; <лог.выр.>; <идент.>=<E>) <спис. операторов> end), Сдвиг
6. Заменить((<лог. выр.>) <спис. операторов><else>), Сдвиг
7. Заменить(<спис. операторов>), Сдвиг
8. Вытолкнуть, Держать
9. Заменить(<F><лог.опер.><F>), Держать
10. Вытолкнуть, Сдвиг
11. Вытолкнуть, Сдвиг
12. Заменить(<T><E-список>), Держать
13. Заменить(<T><E-список>), Сдвиг
14. Вытолкнуть, Держать
15. Заменить(<F><T-список>), Держать
16. Заменить(<F><T-список>), Сдвиг
17. Вытолкнуть, Держать
18. Вытолкнуть, Сдвиг
19. Вытолкнуть, Сдвиг

# - Вытолкнуть, Сдвиг

Д – Допустить

О – Отвергнуть

**Правила замены магазинных символов:**

**Правило 1.** <программа> после

|  |
| --- |
| <оператор> |
| <список операторов> |

до

|  |
| --- |
| <программа> |

**Правило 2.** <список операторов> после

|  |
| --- |
| <оператор> |
| <список операторов> |

до

|  |
| --- |
| <список операторов> |

**Правило 3.** <список операторов> после

до

|  |
| --- |
| <список операторов> |

**Правило 4.** <оператор присваивания> после

|  |  |
| --- | --- |
|  | ID |
| p1 |  |
|  | = |
|  | <E> |
| q1 |  |
|  | {присвоить} |
| p2 |  |
| q2 |  |

до

|  |
| --- |
| <оператор присваивания> |

**Правило 5.** <оператор цикла> после

|  |  |
| --- | --- |
|  | for |
|  | ( |
|  | ID |
| p1 |  |
|  | = |
|  | <E> |
| q1 |  |
|  | {присвоить} |
| p2 |  |
| q2 |  |
|  | ; |
|  | {Метка} |
| v1 | v |
|  | <лог.выр> |
| y1 |  |
|  | {Усл. переход по 0} |
| y2 | (-1) |
| w2 | w |
|  | {Безусловн. Переход} |
| x2 | x |
|  | ; |
|  | {Метка} |
| z1 | z |
|  | ID |
| p3 |  |
|  | = |
|  | <E> |
| q3 |  |
|  | {присвоить} |
| p4 |  |
| q4 |  |
|  | ; |
|  | {Безусловн. Переход} |
| v2 | v |
|  | ) |
|  | {Метка} |
| x1 | x |
|  | <спис. операторов> |
|  | {Безусловн. Переход} |
| z2 | z |
|  | end |
|  | {Метка} |
| w1 | w |

до

|  |
| --- |
| <оператор цикла> |

v:=НовМ

w:=НовМ

x:=НовМ

z:=НовМ

**Правило 6.** <оператор сравнения> после

|  |  |
| --- | --- |
|  | if |
|  | ( |
|  | <лог. выраж.> |
| p1 |  |
|  | ) |
|  | {Усл. переход по 0} |
| p2 |  |
| z1 | z |
|  | <спис. операторов> |
|  | <иначе> |
| z2 | z |
|  | endif |

до

|  |
| --- |
| <оператор сравнения> |

z:=НовМ

**Правило 7.** <иначе> после

|  |  |
| --- | --- |
|  | else |
|  | {Безусловн. Переход} |
| w1 | w |
|  | {метка} |
| z1 | z |
|  | <спис. операторов> |
|  | {метка} |
| w2 | w |

до

|  |  |
| --- | --- |
| <иначе> |  |
| z | z2 |

w:=НовМ

**Правило 8.** <иначе> после

|  |  |
| --- | --- |
|  | {метка} |
| z2 | z |

до

|  |  |
| --- | --- |
| <иначе> |  |
| z | z2 |

**Правило 9.** <лог.выр.> после

|  |  |
| --- | --- |
|  | <F> |
| p1 |  |
|  | <лог.опер.> |
| p2 | p |
| t1 | t |

до

|  |  |
| --- | --- |
| <лог.выр.> |  |
| t | t2 |

**Правило 10.** <лог.опер.> после

|  |  |
| --- | --- |
|  | > |
|  | <F> |
| q1 |  |
|  | {Больше} |
| p2 | p |
| q2 |  |
|  | r1 |

до

|  |  |
| --- | --- |
| <лог.опер.> |  |
| p | p1 |
| t | t2 |

(r1, t2):=НовМ

**Правило 11.** <лог.опер.> после

|  |  |
| --- | --- |
|  | < |
|  | <F> |
| q1 |  |
|  | {Меньше} |
| p2 | p |
| q2 |  |
|  | r1 |

до

|  |  |
| --- | --- |
| <лог.опер.> |  |
| p | p1 |
| t | t2 |

(r1, t2):=НовМ

**Правило 12.** <E> после

|  |  |
| --- | --- |
|  | <T> |
| p1 |  |
|  | <E-список> |
| p2 |  |
| t1 |  |

до

|  |  |
| --- | --- |
| <E> |  |
| t | t2 |

**Правило 13.** <E-список> после

|  |  |
| --- | --- |
|  | + |
|  | <T> |
| q1 |  |
|  | {Сложить} |
|  | p |
| q2 |  |
| r1 | r |
|  | <E-список> |
| r2 |  |
| t1 |  |

до

|  |  |
| --- | --- |
| <E-список> |  |
| p | p1 |
| t | t2 |

r:=Нов

**Правило 14.** <E-список> после

до

|  |  |
| --- | --- |
| <E-список> |  |
| p | p1 |
| t | t2 |

**Правило 15.** <T> после

|  |  |
| --- | --- |
|  | <F> |
| p1 |  |
|  | <T-список> |
| p2 |  |
| t1 |  |

до

|  |  |
| --- | --- |
| <T> |  |
| t | t2 |

**Правило 16.** <T-список> после

|  |  |
| --- | --- |
|  | \* |
|  | <F> |
| q1 |  |
|  | {Умножить} |
|  | p |
| q2 |  |
| r1 | r |
|  | <T-список> |
| r2 |  |
| t1 |  |

до

|  |  |
| --- | --- |
| <T-список> |  |
| p | p1 |
| t | t2 |

r:=Нов

**Правило 17.** <T-список> после

до

|  |  |
| --- | --- |
| <T-список> |  |
| p | p1 |
| t | t2 |

**Правило 18.** <F> после

|  |  |
| --- | --- |
| <ID> |  |
| q | q1 |

до

|  |  |
| --- | --- |
| <F> |  |
| q | q2 |

**Правило 19.** <F> после

|  |  |
| --- | --- |
| <INT> |  |
| q | q1 |

до

|  |  |
| --- | --- |
| <F> |  |
| q | q2 |

**Объектный язык:**

Assign (a, b) – присвоить в a значение b

Add (a, b, c) - присвоить в a значение b+с

Multiply (a, b, c) - присвоить в a значение b\*с

Less (a, b, c) – сравнить a и b, записать в с 1, если результат - «меньше», иначе записать 0

More (a, b, c) - сравнить a и b, записать в с 1, если результат - «больше», иначе записать 0

Unconditional\_jump (a) – переход на метку с адресом a

Conditional\_jump (a, b) – переход по адресу a, если b=1

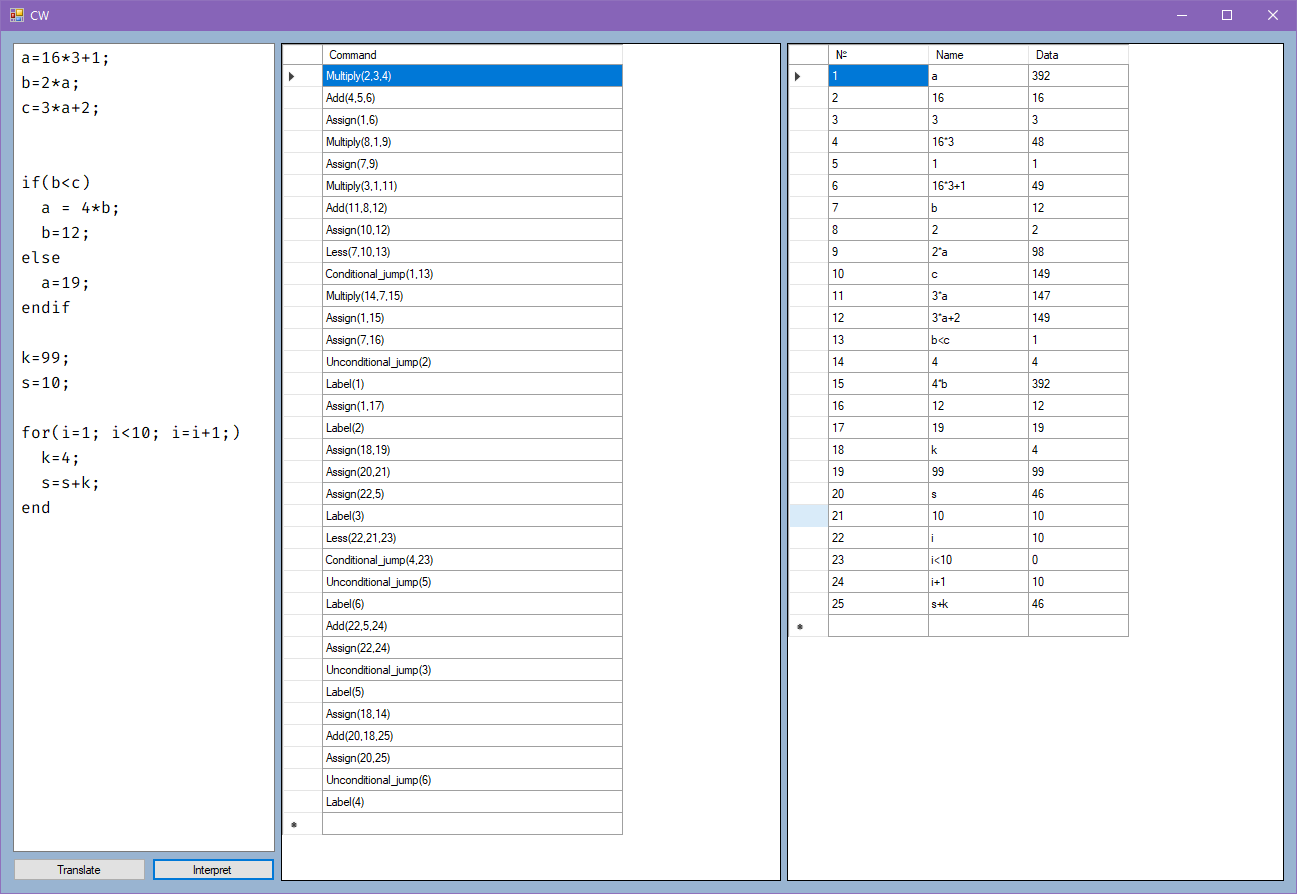
Label(a) – метка с адресом a

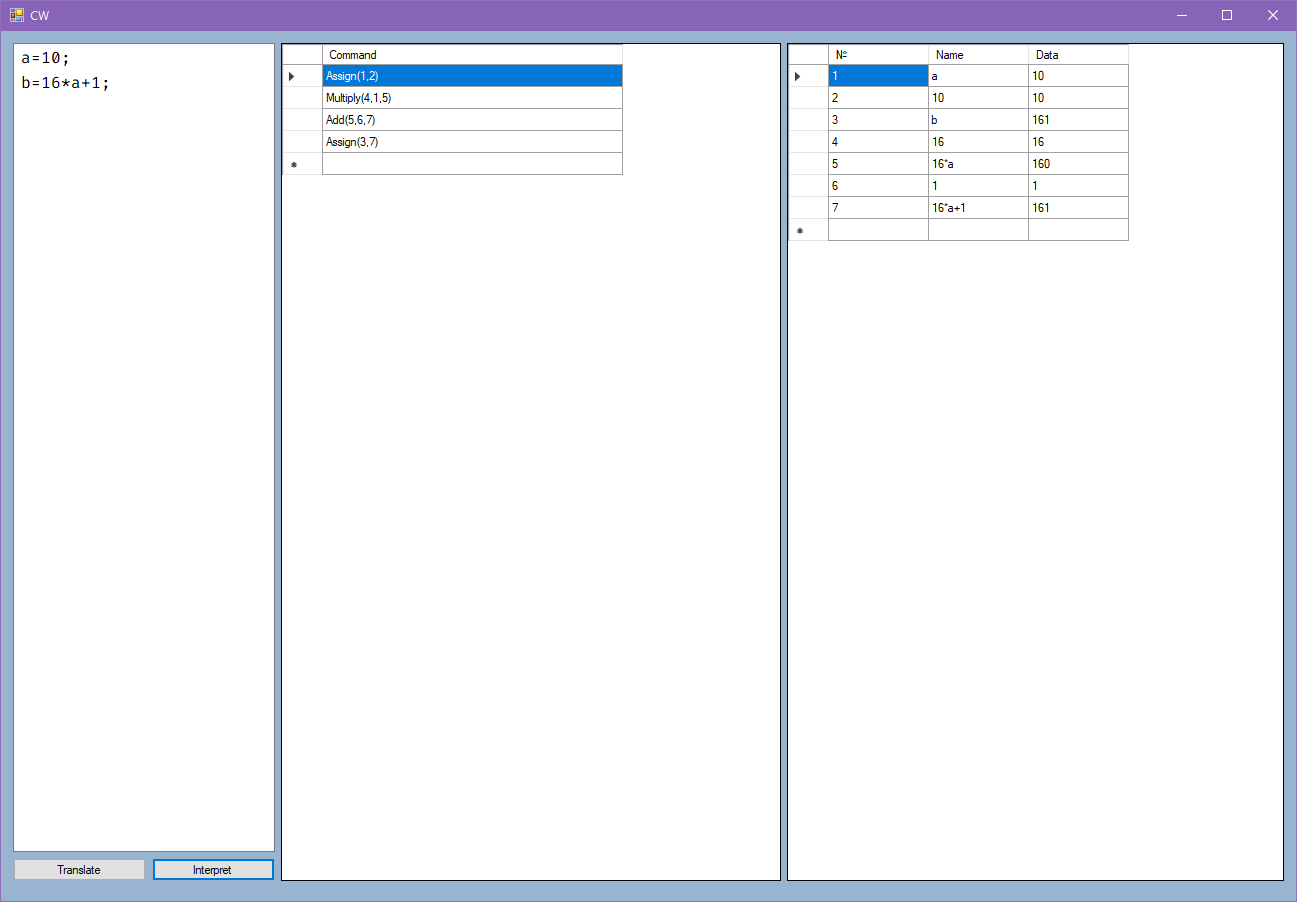
**Описание интерфейса программы:**

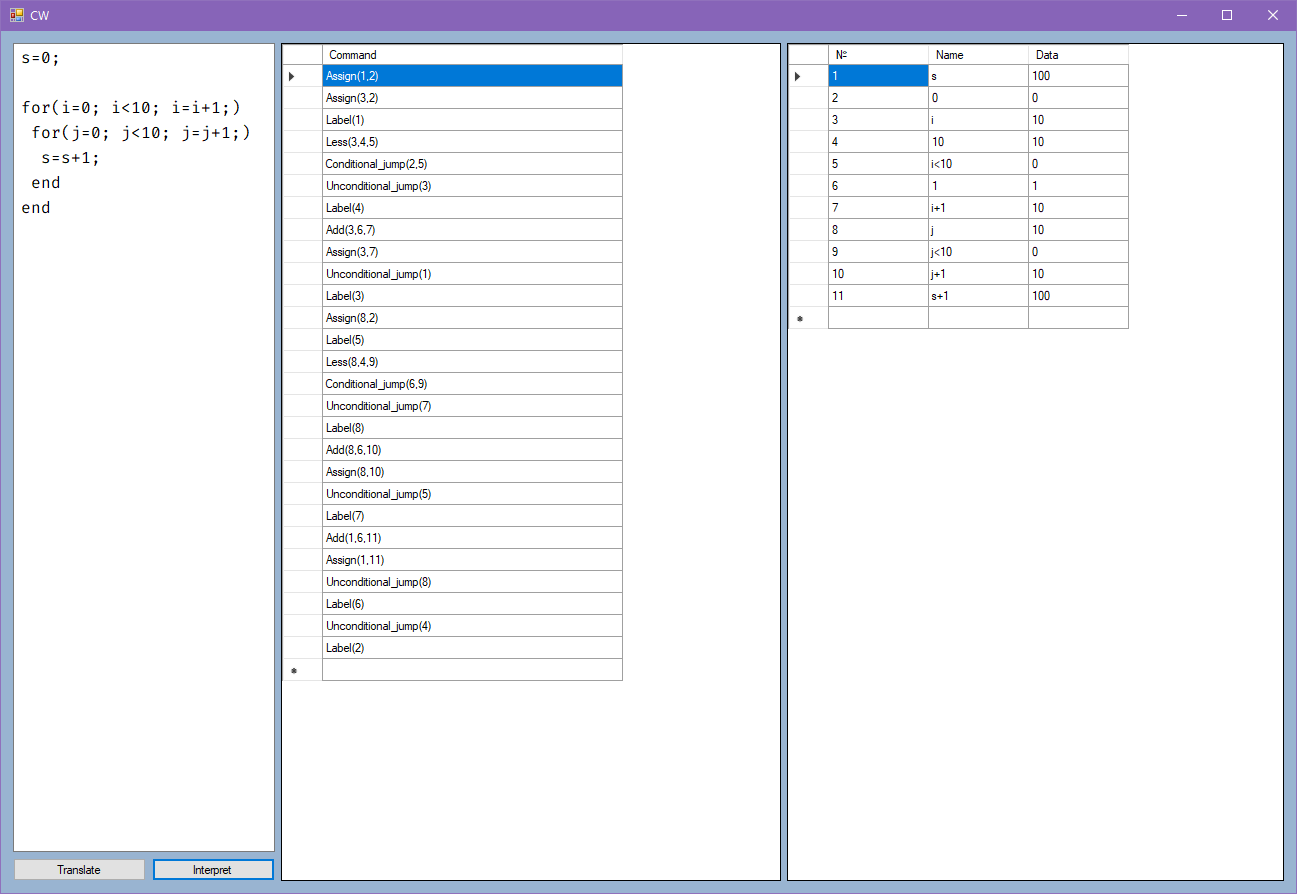
В текстовое поле вводится программа. При нажатии Translate – происходит синтаксический анализ и трансляция. Затем при нажатии Interpret – интерпретация команд объектного языка.

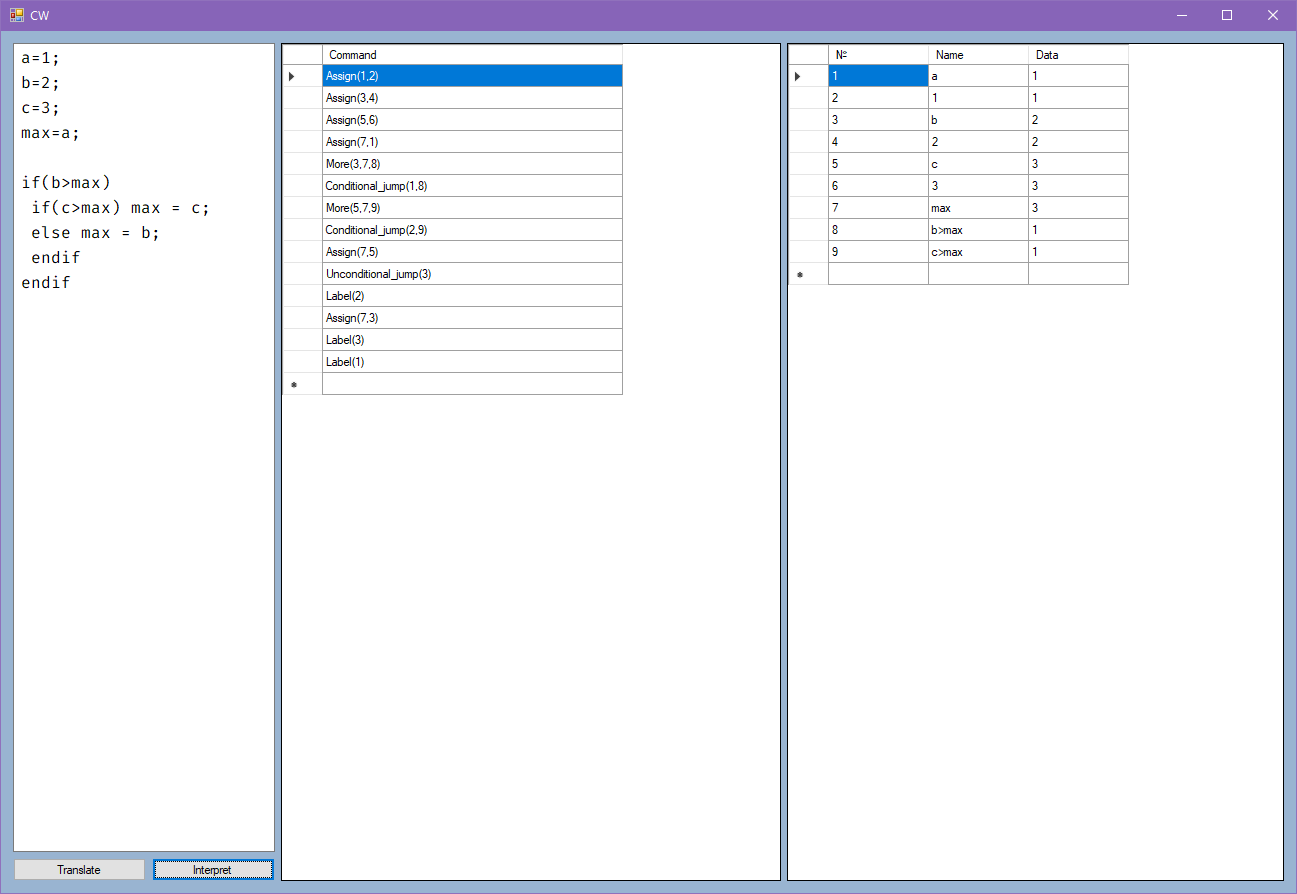
В первой таблице перечислены команды объектного языка для заданной пользователем программы. Во второй таблице – идентификаторы и константы, присутствующие в программе, их адрес и значение

**Тесты работы программы:**

*Тест программы из условия*

*Тест элементарных операций*

*Тест вложенных циклов*

*Тест вложенных сравнений*

**Листинг программы:**

public partial class Form1 : Form

{

Translator translator;

public Form1()

{

InitializeComponent();

}

private void Button1\_Click(object sender, EventArgs e)

{

translator = new Translator("<program>");

translator.Parse(textBox1.Text);

translator.inter.FillGrid(opergrid, datagrid);

}

private void Button2\_Click(object sender, EventArgs e)

{

translator.inter.Execute();

translator.inter.FillGrid(opergrid, datagrid);

}

}

public class Stack<T>

{

class Node

{

public T data;

public Node next;

public Node(T data, Node next = null)

{

this.data = data;

this.next = next;

}

}

public Stack()

{

head = null;

}

Node head;

public void Push(T data)

{

head = new Node(data, head);

}

public T Pop()

{

T data;

if (head != null)

{

data = head.data;

head = head.next;

}

else data = default;

return data;

}

public T Peek()

{

if (head != null)

return head.data;

else return default;

}

}

public class Map<TKey, TValue> where TKey : IComparable where TValue : IComparable

{

class Node

{

public TKey key;

public TValue data;

public Node next;

public Node(TKey key, TValue data, Node next = null)

{

this.key = key;

this.data = data;

this.next = null;

}

}

public Map()

{

head = null;

last = null;

Size = 1;

}

Node head;

Node last;

public int Size { get; set; }

public void Add(TKey key, TValue data)

{

if (Has(key)) return;

if (head == null)

last = head = new Node(key, data, head);

else

last = last.next = new Node(key, data);

Size++;

}

public bool Has(TKey key)

{

Node temp = head;

while (temp != null)

{

if (temp.key.CompareTo(key) == 0)

return true;

temp = temp.next;

}

return false;

}

public TValue this[TKey key]

{

get

{

Node temp = head;

while (temp != null)

{

if (temp.key.CompareTo(key) == 0)

return temp.data;

temp = temp.next;

}

return default;

}

set

{

Node temp = head;

while (temp != null)

{

if (temp.key.CompareTo(key) == 0)

{

temp.data = value;

break;

}

temp = temp.next;

}

}

}

public TKey Get(string l)

{

Node temp = head;

while (temp != null)

{

if (temp.data.CompareTo(l) == 0)

return temp.key;

temp = temp.next;

}

return default;

}

~Map()

{

Node temp = head;

while (head != null)

{

head = head.next;

temp.next = null;

temp = head;

}

}

}

public class Lexemer

{

public Map<int, Lexeme> types;

int Index { get; set; }

public string Input { get; set; }

public Lexemer(string GRAMMAR)

{

types = new Map<int, Lexeme>();

Index = 0;

InitLex(GRAMMAR);

}

public void InitLex(string GRAMMAR)

{

types.Add(types.Size, new Lexeme(types.Size, "INVALID"));

types.Add(types.Size, new Lexeme(types.Size, "EOF"));

var m = new Regex(@"\S+").Matches(GRAMMAR);

for (int i = 0; i < m.Count; i++)

if (m[i].Value != "::=" && m[i].Value != ">>>" && types.Get(m[i].Value) == 0 && m[i].Value != "EOF")

if (m[i].Value.Contains('['))

if (m[i].Value.Contains('{'))

types.Add(types.Size, new Command(types.Size, m[i].Value.Substring(0, m[i].Value.IndexOf('[', 0)), new int[m[i].Value.Split(new[] { ',' }).Length]));

else

types.Add(types.Size, new Lexeme(types.Size, m[i].Value.Substring(0, m[i].Value.IndexOf('[', 0)), new int[m[i].Value.Split(new[] { ',' }).Length]));

else

types.Add(types.Size, new Lexeme(types.Size, m[i].Value));

}

public Lexeme GetLexeme()

{

while (Index != Input.Length && Char.IsWhiteSpace(Input[Index]))

Index++;

if (Index == Input.Length)

return new Lexeme(types.Get("EOF"), "EOF");

if (types.Get(Input.Substring(Index, 1)) != 0 && types.Has(types.Get(Input.Substring(Index, 1))))

{

Index++;

return new Lexeme(types.Get(Input.Substring(Index - 1, 1)), Input.Substring(Index - 1, 1));

}

if (Index != Input.Length - 1 && types.Get(Input.Substring(Index, 2)) != 0 && types.Has(types.Get(Input.Substring(Index, 2))))

{

Index += 2;

return new Lexeme(types.Get(Input.Substring(Index - 2, 2)), Input.Substring(Index - 2, 2));

}

if (!Char.IsDigit(Input[Index]))

{

int start = Index;

while (Index != Input.Length && (Char.IsLetter(Input[Index]) || Char.IsDigit(Input[Index])))

Index++;

if (types.Get(Input.Substring(start, Index - start)) != 0 && types.Has(types.Get(Input.Substring(start, Index - start))))

return new Lexeme(types.Get(Input.Substring(start, Index - start)), Input.Substring(start, Index - start));

else

return new Lexeme(types.Get("<ID>"), Input.Substring(start, Index - start));

}

else

{

int start = Index;

while (Index != Input.Length && (Char.IsLetter(Input[Index]) || Char.IsDigit(Input[Index])))

Index++;

if (int.TryParse(Input.Substring(start, Index - start), out int a))

return new Lexeme(types.Get("<INT>"), Input.Substring(start, Index - start));

else

return new Lexeme(types.Get("INVALID"), Input.Substring(start, Index - start));

}

}

public void Reset()

{

Index = 0;

}

}

public class Var : IComparable

{

public int Data;

public string Name { get; set; }

public bool Set { get; set; }

public Var(string name)

{

Name = name;

Set = false;

}

public Var(string name, int data)

{

Data = data;

Name = name;

Set = true;

}

public int CompareTo(object obj)

{

return Name.CompareTo((string)obj);

}

public override string ToString()

{

return Name;

}

}

public class Lexeme : IComparable

{

public int Type { get; }

public string Name { get; set; }

public int[] attr;

public virtual bool IsCommand()

{

return false;

}

public virtual Lexeme Copy()

{

return new Lexeme(Type, Name, (attr != null) ? new int[attr.Length] : null);

}

public int CompareTo(object obj)

{

return Name.CompareTo((string)obj);

}

public Lexeme(int type, string name, int[] attr = null)

{

Type = type;

Name = name;

this.attr = attr;

}

}

public class Command : Lexeme

{

public Command(int type, string name, int[] attr = null) : base(type, name, attr) { }

public override Lexeme Copy()

{

return new Command(Type, Name, (attr != null) ? new int[attr.Length] : null);

}

public override bool IsCommand()

{

return true;

}

}

class Grammar

{

// MOV - присвоить

// JCC - условный переход

// JMP - безусловный переход

// ADD - сложение

// MUL - умножение

// label - метка

public static string LL1 = @"

<program> ::= <operator> <program> >>> if for <ID>[1]

<program> ::= >>> end endif else EOF

<operator> ::= <assign\_operator> >>> <ID>[1]

<operator> ::= <cond\_operator> >>> if

<operator> ::= <loop\_operator> >>> for

<assign\_operator> ::= <ID>[1] = <E>[2] {MOV}[1,2] ; >>> <ID>[1]

<cond\_operator> ::= if ( <logical\_expression>[1] ) {JCC}[2,1] <program> <else>[2] endif >>> if

<else>[1] ::= {JMP}[2] else {label}[1] <program> {label}[2] >>> else

<else>[1] ::= {label}[1] >>> if for <ID>[1] end endif EOF

<loop\_operator> ::= for ( <assign\_operator> {label}[7] <logical\_expression>[3] {JCC}[8,3] {JMP}[5] ; {label}[6] <assign\_operator> {JMP}[7] ) {label}[5] <program> {JMP}[6] end {label}[8] >>> for

<E>[1] ::= <T>[2] <E\_list>[2,1] >>> <ID>[1] <INT>[1]

<E\_list>[1,2] ::= + <T>[3] {ADD}[1,3,4] <E\_list>[4,2] >>> +

<E\_list>[1,1] ::= >>> ;

<T>[1] ::= <F>[2] <T\_list>[2,1] >>> <ID> <INT>

<T\_list>[1,2] ::= \* <F>[3] {MUL}[1,3,4] <T\_list>[4,2] >>> \*

<T\_list>[1,1] ::= >>> + ;

<F>[1] ::= <ID>[1] >>> <ID>

<F>[1] ::= <INT>[1] >>> <INT>

<logical\_expression>[1] ::= <F>[2] <R\_list>[2,1] >>> <ID> <INT>

<R\_list>[1,2] ::= > <F>[3] {>}[1,3,2] >>> >

<R\_list>[1,2] ::= < <F>[3] {<}[1,3,2] >>> <

";

}

public class Rule

{

public int from;

public int[] to;

public int[] choice;

public int[][] attr;

public Rule(string s, Lexemer lex)

{

string[] temp;

var m = new Regex(@"(\S+)\s\*::=(.\*?)>>>(.\*?)\n").Match(s);

from = lex.types.Get(m.Groups[1].Value.Split(new[] { '[' })[0]);

var m1 = new Regex(@"\S+").Matches(m.Groups[2].Value);

to = new int[m1.Count];

attr = new int[m1.Count + 1][];

if (m.Groups[1].Value.Contains('['))

{

temp = m.Groups[1].Value.Split(new[] { '[', ']', ',' });

attr[0] = new int[temp.Length - 2];

for (int j = 1; j < temp.Length - 1; j++)

attr[0][j - 1] = Int32.Parse(temp[j]);

}

for (int i = 0; i < m1.Count; i++)

if (m1[i].Value.Contains('['))

{

temp = m1[i].Value.Split(new[] { '[', ']', ',' });

attr[i + 1] = new int[temp.Length - 2];

for (int j = 1; j < temp.Length - 1; j++)

attr[i + 1][j - 1] = Int32.Parse(temp[j]);

}

for (int i = 0; i < m1.Count; i++)

to[i] = lex.types.Get(m1[i].Value.Split(new[] { '[' })[0]);

var m2 = new Regex(@"\S+").Matches(m.Groups[3].Value);

choice = new int[m2.Count];

for (int i = 0; i < m2.Count; i++)

choice[i] = lex.types.Get(m2[i].Value.Split(new[] { '[' })[0]);

}

public void ApplyRule(Stack<Lexeme> PDA, Map<int, int> addresses, Lexemer lex)

{

Map<int, int> temp = new Map<int, int>();

Lexeme top = PDA.Peek();

if (attr != null)

{

if (attr[0] != null)

for (int i = 0; i < attr[0].Length; i++)

if (!temp.Has(attr[0][i]))

temp.Add(attr[0][i], top.attr[i]);

else

addresses[top.attr[i]] = addresses[temp[attr[0][i]]];

PDA.Pop();

for (int i = attr.Length - 1; i >= 1; i--)

{

Lexeme t = lex.types[to[i - 1]].Copy();

for (int j = 0; attr[i] != null && j < attr[i].Length; j++)

{

if (temp.Has(attr[i][j]))

t.attr[j] = temp[attr[i][j]];

else

{

temp.Add(attr[i][j], addresses.Size);

addresses.Add(addresses.Size, -1);

t.attr[j] = temp[attr[i][j]];

}

}

PDA.Push(t);

}

}

else PDA.Pop();

}

public Rule(int tfrom)

{

from = tfrom;

choice = new int[1];

to = new int[0];

choice[0] = tfrom;

}

}

public class Translator

{

Rule[,] table;

Lexemer lex;

Stack<Lexeme> PDA;

public Interpretator inter;

public Translator(string start)

{

string GRAMMAR = Grammar.LL1;

lex = new Lexemer(GRAMMAR);

PDA = new Stack<Lexeme>();

inter = new Interpretator();

InitRules(GRAMMAR);

PDA.Push(lex.types[lex.types.Get("EOF")].Copy());

PDA.Push(new Lexeme(lex.types.Get(start), start));

}

void InitRules(string GRAMMAR)

{

table = new Rule[lex.types.Size, lex.types.Size];

var m = new Regex(@"(\S+)\s\*::=(.\*?)>>>(.\*?)\n").Matches(GRAMMAR);

Rule rule;

for (int i = 0; i < m.Count; i++)

{

rule = new Rule(m[i].Value, lex);

for (int j = 0; j < rule.choice.Length; j++)

table[rule.from, rule.choice[j]] = rule;

}

for (int i = 0; i < lex.types.Size; i++)

table[i, i] = new Rule(i);

}

public void PushCommand(Lexeme a)

{

switch (a.Name)

{

case "{MOV}":

a.Name = "Assign";

inter.AddCommand((Command)a);

PDA.Pop();

break;

case "{ADD}":

inter.variables.Add(inter.variables.Size, new Var(inter.variables[inter.addresses[a.attr[0]]].Name + "+" + inter.variables[inter.addresses[a.attr[1]]].Name));

inter.addresses[a.attr[2]] = inter.variables.Size - 1;

inter.AddCommand((Command)a);

a.Name = "Add";

PDA.Pop();

break;

case "{MUL}":

inter.variables.Add(inter.variables.Size, new Var(inter.variables[inter.addresses[a.attr[0]]].Name + "\*" + inter.variables[inter.addresses[a.attr[1]]].Name));

inter.addresses[a.attr[2]] = inter.variables.Size - 1;

a.Name = "Multiply";

inter.AddCommand((Command)a);

PDA.Pop();

break;

case "{JCC}":

if (inter.addresses[a.attr[0]] == -1)

{

inter.labels.Add(a.attr[0], inter.labels.Size);

inter.addresses[a.attr[0]] = inter.labels.Size - 1;

}

a.Name = "Conditional\_jump";

inter.AddCommand((Command)a);

PDA.Pop();

break;

case "{JMP}":

if (inter.addresses[a.attr[0]] == -1)

{

inter.labels.Add(a.attr[0], inter.labels.Size);

inter.addresses[a.attr[0]] = inter.labels.Size - 1;

}

a.Name = "Unconditional\_jump";

inter.AddCommand((Command)a);

PDA.Pop();

break;

case "{>}":

inter.variables.Add(inter.variables.Size, new Var(inter.variables[inter.addresses[a.attr[0]]].Name + ">" + inter.variables[inter.addresses[a.attr[1]]].Name));

inter.addresses[a.attr[2]] = inter.variables.Size - 1;

a.Name = "More";

inter.AddCommand((Command)a);

PDA.Pop();

break;

case "{<}":

inter.variables.Add(inter.variables.Size, new Var(inter.variables[inter.addresses[a.attr[0]]].Name + "<" + inter.variables[inter.addresses[a.attr[1]]].Name));

inter.addresses[a.attr[2]] = inter.variables.Size - 1;

a.Name = "Less";

inter.AddCommand((Command)a);

PDA.Pop();

break;

case "{label}":

if (inter.addresses[a.attr[0]] == -1)

{

inter.labels.Add(a.attr[0], inter.labels.Size);

inter.addresses[a.attr[0]] = inter.labels.Size - 1;

}

a.Name = "Label";

inter.AddCommand((Command)a);

PDA.Pop();

break;

}

}

public bool ParseStep(Lexeme a)

{

int type = a.Type;

int stacktype = PDA.Peek().Type;

do

{

if (PDA.Peek().IsCommand())

{

PushCommand(PDA.Peek());

continue;

}

if (PDA.Peek().Name == "<ID>" && a.Type == lex.types.Get("<ID>"))

{

if ((inter.variables.Size == 0 || inter.variables.Get(a.Name) == 0))

{

inter.variables.Add(inter.variables.Size, new Var(a.Name));

inter.addresses[PDA.Peek().attr[0]] = inter.variables.Size - 1;

}

else

{

inter.addresses[PDA.Peek().attr[0]] = inter.variables.Get(a.Name);

}

}

if (PDA.Peek().Name == "<INT>" && a.Type == lex.types.Get("<INT>"))

{

if ((inter.variables.Size == 0 || inter.variables.Get(a.Name) == 0))

{

inter.variables.Add(inter.variables.Size, new Var(a.Name, Int32.Parse(a.Name)));

inter.addresses[PDA.Peek().attr[0]] = inter.variables.Size - 1;

}

else

{

inter.addresses[PDA.Peek().attr[0]] = inter.variables.Get(a.Name);

}

}

stacktype = PDA.Peek().Type;

Rule rule = table[stacktype, type];

if (rule == null)

{

MessageBox.Show("Expected " + PDA.Peek().Name + " but got " + a.Name);

return false;

}

else

{

rule.ApplyRule(PDA, inter.addresses, lex);

}

} while (stacktype != type);

return true;

}

public void Parse(string input)

{

lex.Input = input;

Lexeme a = null;

do

{

a = lex.GetLexeme();

if (!ParseStep(a))

return;

} while (a.Type != lex.types.Get("EOF"));

MessageBox.Show("Allow");

}

}

public class Interpretator

{

public Command[] PDA;

public Map<int, Var> variables;

public Map<int, int> addresses;

public Map<int, int> labels;

public int size;

public Interpretator()

{

variables = new Map<int, Var>();

addresses = new Map<int, int>();

labels = new Map<int, int>();

PDA = new Command[100];

size = 0;

}

public void AddCommand(Command a)

{

PDA[size] = a;

size++;

}

public void FillGrid(DataGridView oper, DataGridView data)

{

oper.Rows.Clear();

data.Rows.Clear();

oper.ColumnCount = 1;

data.ColumnCount = 3;

oper.Columns[0].Name = "Command";

oper.Columns[0].Width = 300;

data.Columns[0].Name = "№";

data.Columns[1].Name = "Name";

data.Columns[2].Name = "Data";

for (int i = 0; i < size; i++)

{

string attr = "(";

for (int j = 0; j < PDA[i].attr.Length; j++)

if (j == PDA[i].attr.Length - 1)

attr += addresses[PDA[i].attr[j]];

else

attr += addresses[PDA[i].attr[j]].ToString() + ",";

attr += ")";

oper.Rows.Add(PDA[i].Name + attr);

}

for (int i = 1; i < variables.Size; i++)

{

data.Rows.Add(i, variables[i].Name, variables[i].Data);

}

}

public void Execute()

{

int index = 0;

while (index != size)

{

Command com = PDA[index];

switch (com.Name)

{

case "Assign":

if (!variables[addresses[com.attr[1]]].Set)

throw new Exception(String.Format("{0} is not set", variables[addresses[com.attr[1]]].Name));

variables[addresses[com.attr[0]]].Data = variables[addresses[com.attr[1]]].Data;

variables[addresses[com.attr[0]]].Set = true;

index++;

break;

case "Add":

if (!variables[addresses[com.attr[0]]].Set)

throw new Exception(String.Format("{0} is not set", variables[addresses[com.attr[0]]].Name));

if (!variables[addresses[com.attr[1]]].Set)

throw new Exception(String.Format("{0} is not set", variables[addresses[com.attr[1]]].Name));

variables[addresses[com.attr[2]]].Data = variables[addresses[com.attr[0]]].Data + variables[addresses[com.attr[1]]].Data;

variables[addresses[com.attr[2]]].Set = true;

index++;

break;

case "Multiply":

if (!variables[addresses[com.attr[0]]].Set)

throw new Exception(String.Format("{0} is not set", variables[addresses[com.attr[0]]].Name));

if (!variables[addresses[com.attr[1]]].Set)

throw new Exception(String.Format("{0} is not set", variables[addresses[com.attr[1]]].Name));

variables[addresses[com.attr[2]]].Data = variables[addresses[com.attr[0]]].Data \* variables[addresses[com.attr[1]]].Data;

variables[addresses[com.attr[2]]].Set = true;

index++;

break;

case "Conditional\_jump":

if (variables[addresses[com.attr[1]]].Data == 0)

{

for (int i = 0; i < size; i++)

if (PDA[i].Name == "Label" && com.attr[0] == PDA[i].attr[0])

index = i;

}

else

index++;

break;

case "Unconditional\_jump":

for (int i = 0; i < size; i++)

if (PDA[i].Name == "Label" && com.attr[0] == PDA[i].attr[0])

index = i;

break;

case "More":

if (!variables[addresses[com.attr[0]]].Set)

throw new Exception(String.Format("{0} is not set", variables[addresses[com.attr[0]]].Name));

if (!variables[addresses[com.attr[1]]].Set)

throw new Exception(String.Format("{0} is not set", variables[addresses[com.attr[1]]].Name));

if (variables[addresses[com.attr[0]]].Data > variables[addresses[com.attr[1]]].Data)

variables[addresses[com.attr[2]]].Data = 1;

else

variables[addresses[com.attr[2]]].Data = 0;

variables[addresses[com.attr[2]]].Set = true;

index++;

break;

case "Less":

if (!variables[addresses[com.attr[0]]].Set)

throw new Exception(String.Format("{0} is not set", variables[addresses[com.attr[0]]].Name));

if (!variables[addresses[com.attr[1]]].Set)

throw new Exception(String.Format("{0} is not set", variables[addresses[com.attr[1]]].Name));

if (variables[addresses[com.attr[0]]].Data < variables[addresses[com.attr[1]]].Data)

variables[addresses[com.attr[2]]].Data = 1;

else

variables[addresses[com.attr[2]]].Data = 0;

variables[addresses[com.attr[2]]].Set = true;

index++;

break;

case "Label":

index++;

break;

}

}

}

}