Матеріали до виконання лабораторної роботи № 4 Трансляція у ПОЛІЗ операторів розгалуження, циклу та введення-виведення

Юрій Стативка

Квітень, 2020 р.

Зміст

Вступ **2**
Мотиваційний приклад . . . 2
Необхідні програми та дані . . . 3

1 Схема трансляції у ПОЛІЗ **3**
1.1 Мітка та оператор `colon` . . . 3
1.2 Оператор безумовного переходу `goto` . . . 4
1.3 Postfix-інструкція умовного переходу `JF` . . . 6
1.4 Оператор розгалуження `if` . . . 6
1.5 Оператор циклу `for` . . . 8
1.6 Оператори введення-виведення . . . 11
1.7 Загальна схема трансляції . . . 11

2 Програмна реалізація транслятора **12**
2.1 Загальні положення . . . 12
2.2 Код функцій . . . 12
2.2.1 `parseStatement()` . . . 12
2.2.2 `parseIf()` . . . 13
2.2.3 `createLabel()` . . . 14
2.2.4 `setValLabe()` . . . 15
2.2.5 `parseBoolExpr()` . . . 15
2.3 Трансляція програм з розгалуженням . . . 16

3 Про звіт **18**
3.1 Файли та тексти . . . 18
3.2 Форма та структура звіту . . . 18

Література **19**

Вступ

Цей текст підготовлений з огляду на спричинену карантином відсутність лекційних занять як таких. Розглядаються методи трансляції у ПОЛІЗ інструкцій розгалуження `IfStatement`, циклу `ForStatement` і введення-виведення `InpStatement` та `OutStatement` відповідно. Наводиться програмний код розширення транслятора, створеного у попередній лабораторній роботі, для опрацювання інструкцій розгалуження. Код прикладу – додається.

Ті, хто вже зрозуміли як виконати це завдання чи вже виконали його, можуть тільки переглянути вимоги до оформлення звіту у розділі 3 .

Мета лабораторної роботи – програмна реалізація транслятора у ПОЛІЗ усіх синтаксичних конструкцій розробленої мови програмування. Побудова інтерпретатора – завдання наступної лабораторної роботи.

Мотиваційний приклад

Для з'ясування шляхів розширення транслятора у ПОЛІЗ розглянемо приклад програми, написаної певною вхідною мовою:

```
1  program
2      read(a)
3  m2:
4      read(b,h)
5      goto m1
6      h := 2
7  m1: h := 1
8      if a < b then v1 := (a + b)/h
9               else v1 := (a + b)*h
10     endif
11     for i := a+1 step h/2 to (a+b)/2 do
12         h := v1/i
13         write(a+b,h,i)
14     endfor
15 end
```

Бачимо, що необхідно розширити транслятор у ПОЛІЗ для таких конструкцій:

1. Ідентифікатор (оператор) мітки з наступною двокрапкою як точка входження в потоці виконання, рядки 3 – `m2:` та 7 – `m1:`.

2. Інструкція (оператор) безумовного переходу `goto` з наступним ідентифікатором мітки, рядок 5.

3. Інструкція (оператор) розгалуження, вважатимемо її відповідною нетерміналу `IfStatement`, рядки 8 – 10. Зауважимо, що її семантика потребує засобів оцінки значення логічного виразу та умовного переходу.

4. Інструкція (оператор) циклу, вважатимемо, що вона відповідає нетерміналу `ForStatement`, рядки 11 – 14.

5. Інструкції (оператори) введення та виведення значень одного чи списку елементів, рядки 2 – `read(a)`, 4 – `read(b,z)` та 13 – `write(a+b,h,i)`.

Звичайно, вхідна мова не обов'язково містить оператор `goto` та мітки, тож перші два пункти актуальні не для всіх завдань.

Необхідні програми та дані

Для виконання роботи потрібні:

1. Граматика мови.
2. Специфікація розробленої вхідної мови.
3. Лексичний аналізатор.
4. Транслятор для роботи з арифметикою та присвоюванням [3].
5. Приклади програм для тестування транслятора.

Розглянутий далі приклад можна знайти у теці `postfixIF_translator`.

1 Схема трансляції у ПОЛІЗ

Розглянемо можливі підходи до трансляції окремих синтаксичних конструкцій[1], врахувуючи їх семантику та з метою з'ясування загальної схеми[2].

1.1 Мітка та оператор `colon`

Розпізнавання мітки може бути здійснене як на рівні лексичному, так і на синтаксичному за правилом, напр. `Мітка = ІдентМітки ':'` з відповідними семантичними процедурами. Проте бажано зберігати їх у таблиці розбору двома окремими рядками, а `ІдентМітки` заносити до таблиці міток, як це буде показано далі.

При трансляції конструкції `m1:`, якщо цільова мова містить мітки та команди переходу на них, вважаємо, що `:` – оператор, аргументом якого є мітка (ідентифікатор) `m1`. Зауважимо, що він – унарний постфіксний оператор, тобто, як і факторіал, він вже у постфіксній формі,див. Табл. 1, (тут ми не надавали оператору `:` нового позначення):

Семантичні процедури при трансляії оператора `:` такі: 1) додавання мітки до ПОЛІЗу (разом із токеном), 2) додавання до ПОЛІЗу оператора `:` (разом із токеном) і 3) оброблення таблиці міток (додати значення мітки, значення мітки – це номер, під яким вона зберігається перед двокрапкою у ПОЛІЗ-коді).

[1] Пам'ятаємо, що [2] містить приклади трансляції великої кількості різноманітних синтаксичних конструкцій

[2] Про термінологію див. *Зауваження* на стор. 8

№	Оператор	Вираз	Постфіксна форма
1	Постфіксний	5!	5 FACT
2	Постфіксний	m1 :	m1 :

Табл. 1: Приклади запису постфіксних операторів

Наприклад, розбір та трансляція програми

```
1 program
2     m5:
3     m2:
4 end
```

дають такий результат:

```
 Таблиця символів
numRec     numLine    lexeme     token      index
1          1          program    keyword
2          2          m5         label
3          2          :          colon
4          3          m2         label
5          3          :          colon
6          4          end        keyword

 Таблиця ідентифікаторів
Ident      Type            Value           Index

 Таблиця констант
Const      Type       Value      Index

 Таблиця міток
Label      Value
m5         0
m2         2

Код програми у постфіксній формі (ПОЛІЗ):
[('m5', 'label'), (':', 'colon'), ('m2', 'label'), (':', 'colon')]
```

1.2 Оператор безумовного переходу `goto`

Унарний префіксний оператор у постфікс–коді теж записується після свого аргументу, як у Табл. 2:

На рівні інструкцій ПОЛІЗу застосуємо лексему `JUMP` з токеном `jump`. Розробник вільний у виборі відповідних лексем, так у [2] використовується `БП` – від **Б**езумовний **П**ерехід.Для нас важливо, що лексеми `JUMP` та `goto` мають тотожну семантику, але у мовах проміжного та високого рівня відповідно, і при використанні не потребують додаткових уточнень.

№	Оператор	Вираз	Постфіксна форма
1	Префіксний	-9	9 NEG
2	Префіксний	goto m1	m1 JUMP

Табл. 2: Приклади запису префіксних операторів

Семантичні процедури при трансляії оператора `goto` зводяться до запису у таблицю розбору та внесення до ПОЛІЗ-коду.

У мовах з динамічною типізацією лексичний аналізатор, позбавлений зв'язку з парсером, може обробити ідентифікатор мітки як ідентифікатор та зробити відповідні записи у таблиці розбору та таблиці ідентифікаторів (а не таблиці міток). У такому випадку вказані недоліки мають бути скориговані на етапі синтаксичного розбору. Наступний приклад демонстує таку ситуацію для програми

```
1 program
2   m2:
3     goto m1
4     goto m2
5   m1:
6 end
```

```
 Таблиця символів
numRec     numLine    lexeme     token      index
1          1          program    keyword
2          2          m2         label
3          2          :          colon
4          3          goto       keyword
5          3          m1         ident      1
6          4          goto       keyword
7          4          m2         ident      2
8          5          m1         label
9          5          :          colon
10         6          end        keyword

 Таблиця ідентифікаторів
Ident      Type            Value           Index
m1         type_undef      val_undef       1
m2         type_undef      val_undef       2

 Таблиця констант
Const      Type       Value      Index
```

```
 Таблиця міток
Label      Value
m2         0
m1         6

Код програми у постфіксній формі (ПОЛІЗ):
[('m2', 'label'), (':', 'colon'), ('m1', 'ident'), ('JUMP', 'jump'),
 ('m2', 'ident'), ('JUMP', 'jump'), ('m1', 'label'), (':', 'colon')]
```

В такій ситуації, після лексичного аналізу, можна піти одним зі шляхів:

1. видалити з таблиці ідентифікаторів записи з лексемами ідентифікаторами-мітками; в таблиці розбору замінити записи типу `5:(3, 'm1', 'ident',1` на `5:(3, 'm1', 'label',''`.

2. Видалити з таблиці ідентифікаторів записи з лексемами ідентифікаторами-мітками; в таблиці розбору залишити записи як є, але при інтерпретації `('JUMP','jump')` ігнорувати токен 'ident' лексеми на вершині стека.

3. Запропонувати інший спосіб.

1.3 Postfix-інструкція умовного переходу JF

Мови проміжного та низького рівня містять команди умовного переходу на мітку. Тут буде використовуватись постфіксна команда у формі
`BoolExpr Label JF`, яка здійснює перехід на мітку `Label` тільки у випадку, якщо `BoolExpr` має значення `false`. У формі таблиці фрагмент postfix-коду можна представити так (нижній рядок – адреси/номери елементів програми у постфіксній нотації): Якщо в результаті виконання інструкції `<`, номер 2, на

a	x	$<$	m_1	JF	...	...	m_1	:	...
0	1	2	3	4	5	...	n	n+1	n+2

стек буде покладено `false`, то команда `JF` здійснить перехід по мітці m_1, тобто наступною встановить команду з номером `n`. Інакше команда `JF` наступною встановить інструкцію з номером 5.

1.4 Оператор розгалуження if

Оператор розгалуження у граматиці представлений правилом для нетермінала `IfStatement`

```
IfStatement = if BoolExpression
                    then StatementList1
                    else StatementList2
              endif
```

Для зручності графічного представлення перепишемо його з очевидними позначеннями (скороченими)

```
IfStatement = if BoolExpr then SL1 else SL2 endif
```

Для з'ясування способу трансляції у ПОЛІЗ розглянемо блок-схему оператора та розставимо мітки, наприклад так, як на Рис. 1. Семантика оператора очевидна:

1. Якщо `BoolExpr` має значення `False`, то перейти на мітку m_1, потім виконати `SL2`, потім перейти на m_2.

2. Якщо `BoolExpr` має значення `True`, то виконати `SL1`, потім перейти на m_2.

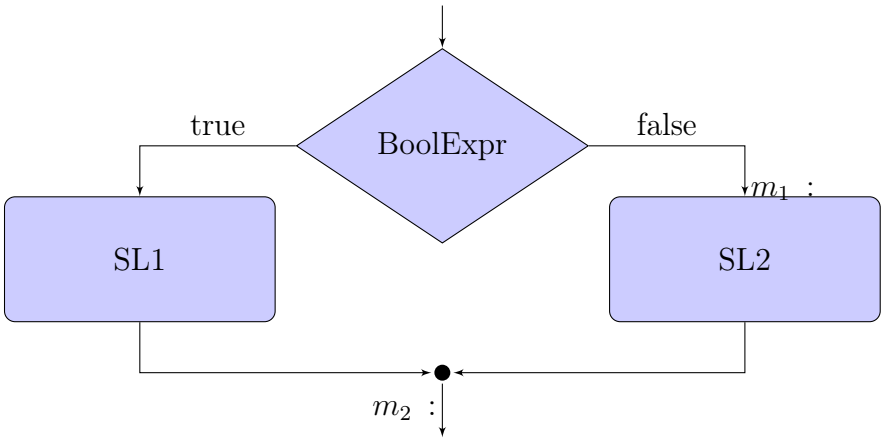

Рис. 1: Блок-схема оператора if

За допомогою ПОЛІЗ-інструкцій умовного та безумовного переходу схематично це можна представити, як у Табл. 3.

ПОЛІЗ(BoolExpr)		m_1	JF	SL1	m_2	JUMP	m_1	:	SL2	m_2	:	...
...	n	n+1	n+2	n+3	n+4	n+5	n+6	n+7	n+8	n+9	n+10	n+11

Табл. 3: Схема постфікс-коду оператора `if`

Отже, якщо `BoolExpr` має значення `false`, то за інструкцією `JF` перейти на мітку m_1 з номером `n+6` (виконання інструкцій `n+6` та `n+7` означають просто перехід до інструкції з номером, на одиницю більшим , тобто до `n+7` та `n+8` відповідно), потім виконати `SL2`, далі – мітка m_2.

Якщо ж `BoolExpr` має значення `true`, то інструкція `JF` не спрацьовує, далі виконується `SL1`, потім безумовний перехід `JUMP`, інструкція `n+5`, передає управління на m_2 – інструкцію з номером `n+9`.

З порівняння структури оператора `if` вхідною та постфіксною мовами отримаємо схему трансляції, див. Рис. 2.

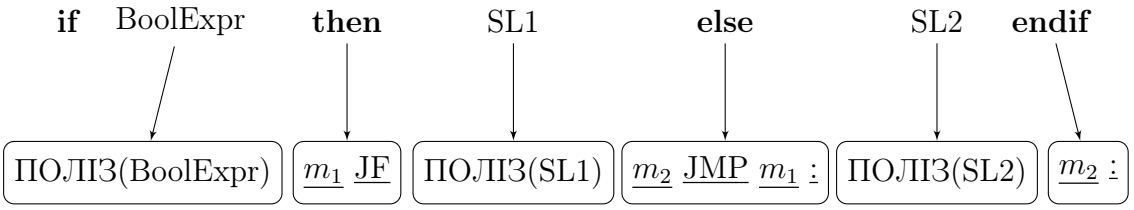

Рис. 2: Схема трансляції оператора `if`

Іншими словами, **схема трансляції** розглянутого тут оператора `if` полягає в таких діях:

1. Нічого не робити, коли зустрілась лексема `if`.
2. Виконати трансляцію у постфіксну форму логічного виразу `BoolExpr`.
3. Зустрівши лексему `then` – згенерувати нову мітку, хай це буде m_1; додати до ПОЛІЗу m_1 `JF`.
4. Виконати трансляцію у постфіксну форму списку операторів `SL1`.
5. Зустрівши лексему `else` – згенерувати нову мітку, хай це буде m_2; додати до ПОЛІЗу m_2 `JMP` m_1 `:`.
   Виконати семантичні процедури оператора `colon`, див. розділ 1.1.
6. Виконати трансляцію у постфіксну форму списку операторів `SL2`.
7. Зустрівши лексему `endif` – додати до ПОЛІЗу m_2 `:`.
   Виконати семантичні процедури оператора `colon`, див. розділ 1.1.

Зауваження. Схема трансляції (синтаксично керована) визначається в [1, розд. 2.3 та 5.4] як граматика з прикріпленими до продукцій (правил) програмних фрагментів. Самі фрагменти називаються також діями, а у випадку атрибутивних граматик – семантичними правилами.
В цьому тексті термін схема трансляції означає також графічне та/або словесне представлення процесу аналізу–синтезу, який забезпечує трансляцію з вхідної мови на цільову (ПОЛІЗ). Такі терміни як дія, семантична процедура, семантичне правило тощо вважаються тут синонімами.

1.5 Оператор циклу `for`

Розглянемо трансляцію оператора циклу такої структури:

```
for Prm := StartExpr step StepExpr to TargExpr do
   StatementList
endfor
```

Вважаючи позначення очевидними, розглянемо його семантику:

1. На першій ітерації параметру циклу `Prm` присвоїти значення виразу `StartExpr`.

2. Обчислити значення `StepExpr`. Але якщо це перша ітерація, то ігнорувати його, інакше збільшити параметр циклу `Prm` на значення кроку `StepExpr`, тобто `Prm := Prm + StartExpr`.

3. Перевірити, що значення параметра циклу знаходиться в інтервалі між `StartExpr` та `TargExpr`. Якщо це так, то виконати тіло циклу `StatementList` та перейти до обчислення `StepExpr`, п. 2, інакше вийти з циклу.

Позначивши ознаку першої/непершої ітерації як `r1`, значення 1 і 0 відповідно, та змінну для зберігання значення `StepExpr` як `r2`, побудуємо блок-схему, встановивши мітки для переходів, див. Рис. 3.

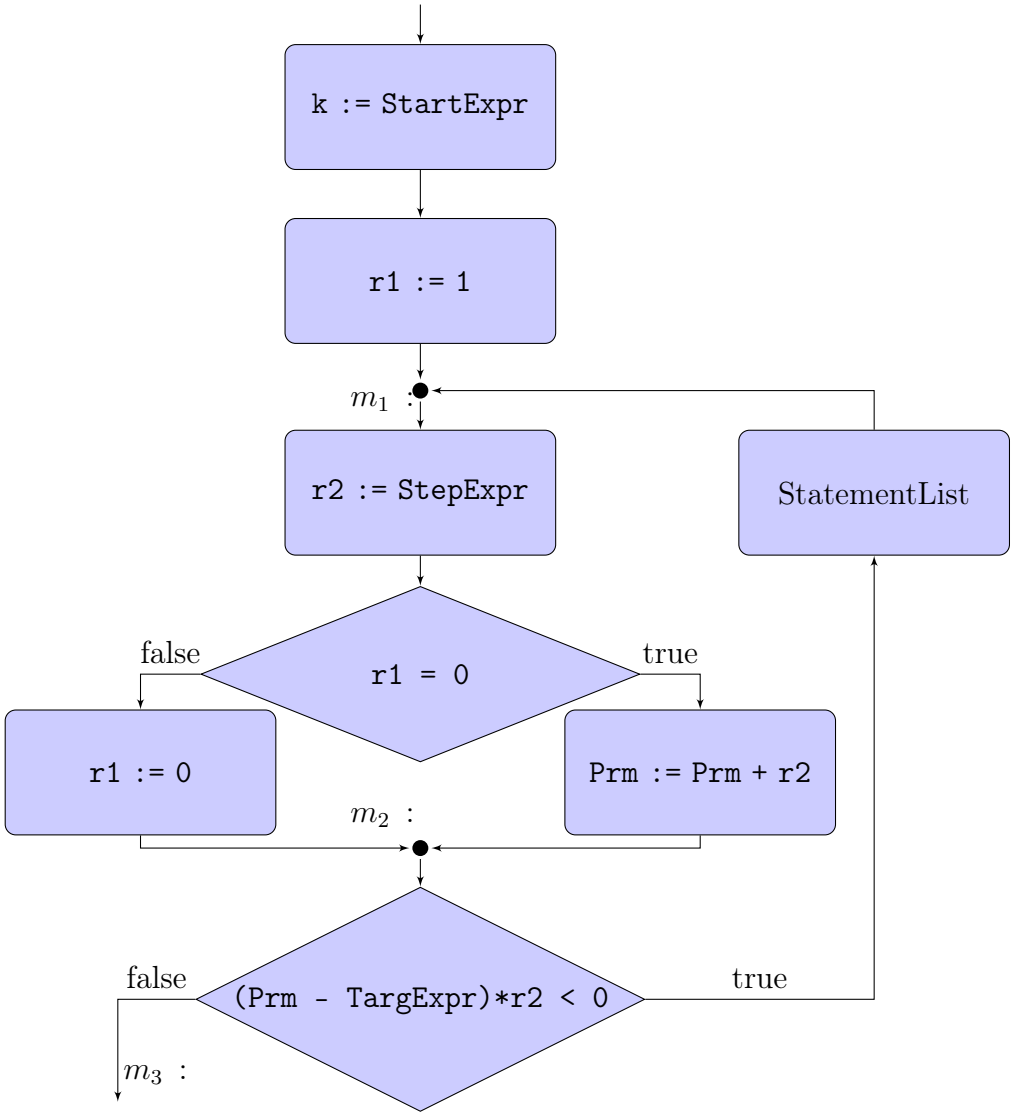

Рис. 3: Блок-схема інструкції if

Тоді схема трансляції та відповідні семантичні процедури, можуть бути представлені, див. також Табл. 4, так:

№	Елемент	Переклад
1	`for`	
2	`Prm := StartExpr`	`Prm ПОЛІЗ(StartExpr) :=`
3	`step`	r_1 `1 :=` m_1 `:` r_2
4	`StepExpr`	`ПОЛІЗ(StepExpr)`
5	`to`	`:=` r_1 `0 =` m_2 `JF Prm Prm` r_2 `+ :=` m_2 `:` r_1 `0 :=Prm`
6	`TargExpr`	`ПОЛІЗ(TargExpr)`
7	`do`	`-` r_2 `* 0 ≤` m_3 `JF`
8	`StatementList`	`ПОЛІЗ(StatementList)`
9	`endfor`	m_1 `JUMP` m_3 `:`

Табл. 4: Схема трансляції оператора `for`

1. Нічого не робити, коли зустрілась лексема `for`.
2. Додати до ПОЛІЗ-коду три елементи – `Prm ПОЛІЗ(StartExpr) :=`.
3. Зустрівши лексему `step` – згенерувати нову мітку, хай це буде m_1; створити, якщо вони ще не створені, дві внутрішні змінні з зарезервованими ідентифікаторами r_1 та r_2; додати до ПОЛІЗу інструкції r_1 `1 :=` m_1 `:` r_2.
4. Додати до ПОЛІЗ-коду `ПОЛІЗ(StepExpr)`.
5. Зустрівши лексему `to` – згенерувати нову мітку m_2; додати до ПОЛІЗ-коду інструкції `:=` r_1 `0 =` m_2 `JF Prm Prm` r_2 `+ :=` m_2 `:` r_1 `0 :=Prm`.
6. Додати до ПОЛІЗ-коду `ПОЛІЗ(TargetExpr)`.
7. Зустрівши лексему `do` – згенерувати нову мітку m_3; додати до ПОЛІЗ-коду інструкції `-` r_2 `* 0 ≤` m_3 `JF`.
8. Додати до ПОЛІЗ-коду `ПОЛІЗ(StatementList)`.
9. Зустрівши лексему `endfor` – додати до ПОЛІЗ-коду інструкції m_1 `JUMP` m_3 `:`

Так, наприклад коду вхідною мовою

```
for i := a+1 step h/2 to (a+b)/2 do
    h := v1/i
    write(a+b,h,i)
endfor
```

відповідатиме постфіксний код:

`i a 1 + :=` r_1 `1 :=` m_1 `:` r_2 `h 2 / :=` r_1 `0 =` m_2 `JF i i` r_2 `+ :=` m_2 `:` r_1 `0 := i a b + 2 / - ` r_2 `* 0 ≤` m_3 `JF h v1 i / := a b + OUT h OUT i OUT` m_1 `JUMP` m_3 `:`

Або у формі Табл. 5.

№	Елемент	Переклад
1	`for`	
2	`i := a+1`	i a 1 + :=
3	`step`	r_1 1 := m_1 : r_2
4	`h / 2`	h 2 /
5	`to`	:= r_1 0 = m_2 JF i i r_2 + := m_2 : r_1 0 := i
6	`(a + b) / 2`	a b + 2 /
7	`do`	- r_2 * 0 $\leq$ m_3 JF
8	`h := v1/i` `write(a+b,h,i)`	h v1 i / := a b + OUT h OUT i OUT
9	`endfor`	m_1 JUMP m_3 :

Табл. 5: Приклад трансляції оператора `for`

1.6 Оператори введення-виведення

Підхі до трансляції операторів введення-виведення розглянемо на прикладі нетермінала `Out`, див. також приклад у рядку 8 Табл. 5:

```
Out = write '(' OutExprList ')'
OutExprList = OutExpr  { ',' OutExpr }
```

Представимо правила для `Out` у формі:

```
Out = write '('  OutExpr  { ',' OutExpr } ')'
```

Схема трансляції `Out` може бути такою:

1. Нічого не робити, коли зустрілись лексеми `write` та `(`.
2. Зустрівши лексему `,` (кома) або `)` – додати до ПОЛІЗу інструкцію `OUT`, інакше – транслювати у ПОЛІЗ `OutExpr`.

1.7 Загальна схема трансляції

Розглянуті приклади свідчать, що розробка схеми трансляції довільної синтаксичної конструкції вхідної мови складається з таких кроків:

1. Розглянути правила граматики.
2. Описати (специфікація мови) семантику конструкції.
3. Для складних конструкцій – побудувати блок-схему з використанням міток.
4. Описати блок-схему (у простих випадках – правила граматики) інструкціями постфікс-коду (з використанням міток та інструкцій умовного/безумовного переходу), записуючи `ПОЛІЗ(`X_i`)` замість високорівневих конструкцій X_i.

5. Із зіставлення правил граматики та опису з попереднього пункту встановити відповідність між високорівневими конструкціями X_i та ПОЛІЗ(X_i), а решту фрагментів постфікс-коду асоціювати з терміналами правил граматики.

6. Додати до отриманої схеми опис інших необхідних дій (семантичних процедур), таких як, наприклад, додавання мітки до таблиці міток при її створенні.

2 Програмна реалізація транслятора

2.1 Загальні положення

Розроблений раніше транслятор для арифметики вхідної мови тут розширюється до опрацювання оператора розгалуження.

Таблиця символів мови доповнена логічними константами, у лексичному аналізаторі `tableOfLanguageTokens` – лексемами/токенами `'true':'bool'` та `'false':'bool'`. Синтаксичний аналізатор/транслятор доповнений функцією для розбору та трансляції логічних виразів за правилом:

```
BoolExpr = true
         | false
         | Expression ('='|'<='|'>='|'<'|'>'|'<>') Expression
```

де `Expression` означає арифметичний вираз.

Розширення транслятора змістовно зводиться до розширення функції `parseStatement()`, яка тепер крім оператора присвоювання обробляє ще і оператор розгалуження.

Інтерпретатор для виконання програм з оператором розгалуження тут не розглядається.

2.2 Код функцій

2.2.1 `parseStatement()`

Ця функція доповнена викликом `parseIf()`, рядки 13 – 15, для розбору та трансляції оператора розгалуження та викликом у рядках 17 – 22 функцій, реалізація яких у цьому прикладі не представлена.

```
1  def parseStatement():
2      # print('\t\t parseStatement():')
3      # прочитаємо поточну лексему в таблиці розбору
4      numLine, lex, tok = getSymb()
5      # якщо токен - ідентифікатор
6      # обробити інструкцію присвоювання
7      if tok == 'ident':
8          parseAssign()
9          return True
10
```

```
11      # якщо лексема - ключове слово 'if'
12      # обробити інструкцію розгалуження
13      elif (lex, tok) == ('if','keyword'):
14          parseIf()
15          return True
16
17      elif tok == 'label':
18          parseLabel()                        # stub
19          return True
20      elif (lex, tok) == ('goto','keyword'):
21          parseGoto()                         # stub
22          return True
23
24      # тут - ознака того, що всі інструкції були коректно
25      # розібрані і була знайдена остання лексема програми.
26      # тому parseStatement() має завершити роботу
27      elif (lex, tok) == ('end','keyword'):
28              return False
29
30      else:
31          # жодна з інструкцій не відповідає
32          # поточній лексемі у таблиці розбору,
33          failParse('невідповідність інструкцій',(numLine,lex,tok,'ident або if'))
34          return False
```

2.2.2 parseIf()

Тут реалізована схема трансляції, див. розділ 1.4. Зіставлення див. у Табл 6

```
1  # розбір інструкції розгалуження за правилом
2  # IfStatement = if BoolExpr then Statement else Statement endif
3  # функція названа parseIf() замість parseIfStatement()
4  def parseIf():
5      global numRow
6      _, lex, tok = getSymb()
7      if lex=='if' and tok=='keyword':
8          # 'if' нічого не додає до ПОЛІЗу    # Трансляція
9          numRow += 1
10         parseBoolExpr()                     # Трансляція
11         parseToken('then','keyword','\t'*5)
12         # Згенерувати мітку m1 = (lex,'label')
13         m1 = createLabel()
14         postfixCode.append(m1)  # Трансляція
15         postfixCode.append(('JF','jf'))
16                                 # додали m1 JF
17
18         parseStatement()        # Трансляція
```

```
19
20          parseToken('else','keyword','\t'*5)
21          # Згенерувати мітку m2 = (lex,'label')
22          m2 = createLabel()
23          postfixCode.append(m2)  # Трансляція
24          postfixCode.append(('JMP','jump'))
25          setValLabel(m1) # в табл. міток
26          postfixCode.append(m1)
27          postfixCode.append((':','colon'))
28                                  # додали m2 JMP m1 :
29          parseStatement()        # Трансляція
30          parseToken('endif','keyword','\t'*5)
31          setValLabel(m2) # в табл. міток
32          postfixCode.append(m2)  # Трансляція
33          postfixCode.append((':','colon'))
34                                  # додали m2 JMP m1 :
35          return True
36      else: return False
```

№	Дія схеми трансляції	Рядки
1	Нічого не робити, коли зустрілась лексема `if`	9
2	Виконати трансляцію логічного виразу `BoolExpr` у постфіксну форму	11
3	Зустрівши лексему `then` – згенерувати нову мітку, хай це буде m_1; додати до ПОЛІЗу m_1 `JF`	12–16
4	Виконати трансляцію у постфіксну форму списку операторів `SL1`	19
5	Зустрівши лексему `else` – згенерувати нову мітку, хай це буде m_2; додати до ПОЛІЗу m_2 `JMP` m_1 `:`.Виконати семантичні процедури оператора `colon`, див. розділ 1.1	21–29
6	Виконати трансляцію у постфіксну форму списку операторів `SL2`	30
7	Зустрівши лексему `endif` – додати до ПОЛІЗу m_2 `:`. Виконати семантичні процедури оператора `colon`, див. розділ 1.1	31–34

Табл. 6: Відповідність коду схемі трансляції оператора `if`

2.2.3 `createLabel()`

Програмно згенеровані мітки отримують ідентифікатори `'m'+наступнийВільнийНомер`, рядки 3–4. Якщо мітки з таким ідентифікатором немає у таблиці міток – додаємо її до таблиці міток з невизначеним значенням, рядки 5–7, інакше виконання програми переривається з виведенням повідомлення `'Конфлікт міток'`, рядки 9–12. Зауважимо, що можна було би шукати справді `наступнийВільнийНомер` у циклі, не обмежуючись номером `len(tableOfLabel)+1`, – тоді конфлікт імен міток не виникав би.

```
1  def createLabel():
2      global tableOfLabel
3      nmb = len(tableOfLabel)+1
4      lexeme = "m"+str(nmb)
5      val = tableOfLabel.get(lexeme)
6      if val is None:
7          tableOfLabel[lexeme] = 'val_undef'
8          tok = 'label' # # #
9      else:
10         tok = 'Конфлікт міток'
11         print(tok)
12     exit(1003)
13      return (lexeme,tok)
```

2.2.4 setValLabe()

Тут встановлюється значення мітки, у відповідності до семантичних процедур оператора colon, див. розділ 1.1. У рядку 4 номер запису у постфікс-коді, куди буде записана мітка, встановлюється як значення цієї мітки .

```
1  def setValLabel(lbl):
2      global tableOfLabel
3      lex,_tok = lbl
4      tableOfLabel[lex] = len(postfixCode)
5      return True
```

2.2.5 parseBoolExpr()

Синтаксичний аналізатор/транслятор доповнений функцією для розбору та трансляції логічних виразів за правилом:

```
BoolExpr = true
         | false
         | Expression ('='|'<='|'>='|'<'|'>'|'<>') Expression
```

де Expression означає арифметичний вираз.

```
1  def parseBoolExpr():
2      global numRow
3      numLine, lex, tok = getSymb()
4      if lex == 'true' or lex == 'false':
5          numRow += 1
6          postfixCode.append((lex,tok))   # Трансляція
7          return True
8      else:
9          parseExpression()               # Трансляція
10         numLine, lex, tok = getSymb()
11         numRow += 1
```

```
12          parseExpression()                # Трансляція
13      if tok in ('rel_op'):
14          postfixCode.append((lex,tok))    # Трансляція
15          # print('\t'*5+'в рядку {0} - {1}'.format(numLine,(lex, tok)))
16      else:
17          failParse('mismatch in BoolExpr',(numLine,lex,tok,'relop'))
18      return True
```

2.3 Трансляція програм з розгалуженням

Переглянемо процес трансляції програми

```
1 program
2 if a + b < c then s := 10 else d := 100 endif
3 end
```

Запуск транслятора та результати його роботи виглядають у консолі приблизно так:

```
>python postfixIF_translator.py
Translator: Переклад у ПОЛІЗ та синтаксичний аналіз завершились успішно

 Таблиця міток
Label      Value
m1         12
m2         17

 Таблиця ідентифікаторів
Ident      Type            Value           Index
a          type_undef      val_undef       1
b          type_undef      val_undef       2
c          type_undef      val_undef       3
s          type_undef      val_undef       4
d          type_undef      val_undef       5

Початковий код програми:
program
if a + b < c then s := 10 else d := 100 endif
end

Код програми у постфіксній формі (ПОЛІЗ):
[('a', 'ident'), ('b', 'ident'), ('+', 'add_op'), ('c', 'ident'),
 ('<', 'rel_op'), ('m1', 'label'), ('JF', 'jf'), ('s', 'ident'),
 ('10', 'int'), (':=', 'assign_op'), ('m2', 'label'), ('JMP', 'jump'),
 ('m1', 'label'), (':', 'colon'), ('d', 'ident'), ('100', 'int'),
 (':=', 'assign_op'), ('m2', 'label'), (':', 'colon')]
```

Трансляція виконана правильно. Справді, всі змінні створені, але їх значення та типи не визначені із-за відсутності відповідних команд у вхідній програмі.

Перехід за міткою m_1 здійснюється на інструкцію постфіксного коду з номером 12, тобто, врешті, до присвоювання значення 100 змінній `d`. Якщо ж інструкція `JF` не спрацьовує (тобто, якщо на момент її виконання на вершині стека –лексема `true`), то виконується наступний за `JF` код – присвоєння значення 10 змінній з ідентифікатором `s` та безумовний перехід на мітку m_2. Виконаємо трансляцію програми з вкладеними операторами

```
1  program
2  if a + b < c
3      then
4          if a = b
5              then s := 5
6              else d := 500
7          endif
8      else d := 100
9  endif
10 end
```

Результати виглядають у консолі приблизно так:

```
>python postfixIF_translator.py
Translator: Переклад у ПОЛІЗ та синтаксичний аналіз завершились успішно

 Таблиця міток
Label      Value
m1         26
m2         17
m3         22
m4         31

 Таблиця ідентифікаторів
Ident      Type           Value          Index
a          type_undef     val_undef      1
b          type_undef     val_undef      2
c          type_undef     val_undef      3
s          type_undef     val_undef      4
d          type_undef     val_undef      5

Код програми у постфіксній формі (ПОЛІЗ):
[('a', 'ident'), ('b', 'ident'), ('+', 'add_op'), ('c', 'ident'),
 ('<', 'rel_op'), ('m1', 'label'), ('JF', 'jf'), ('a', 'ident'),
 ('b', 'ident'), ('=', 'rel_op'), ('m2', 'label'), ('JF', 'jf'),
 ('s', 'ident'), ('5', 'int'), (':=', 'assign_op'), ('m3', 'label'),
```

```
('JMP', 'jump'), ('m2', 'label'), (':', 'colon'), ('d', 'ident'),
('500', 'int'), (':=', 'assign_op'), ('m3', 'label'), (':', 'colon'),
('m4', 'label'), ('JMP', 'jump'), ('m1', 'label'), (':', 'colon'),
('d', 'ident'), ('100', 'int'), (':=', 'assign_op'), ('m4', 'label'),
(':', 'colon')]
```

3 Про звіт

3.1 Файли та тексти

Нагадую, що звіти про виконання лабораторних робіт треба надсилати у форматі pdf. Називати файли треба за шаблоном
`НомерГрупи.ЛР_Номер.ПрізвищеІніціали.pdf`, наприклад так
`ТВ-71.ЛР_3.АндрієнкоБВ.pdf`.

Тут *дуже важливо, щоб дефіс, підкреслювання та крапка були саме на своїх місцях, не було зайвих пробілів чи інших символів*. Одноманітність назв значно зменшує трудомісткість перевірки та імовірність помилки при обліку виконаних вами робіт.

Разом з *вільним розповсюдженням цих матеріалів* (поштою, месенджерами тощо), прошу *не розміщувати їх у вільному доступі*. Це позбавить мене зайвих клопотів при їх офіційному виданні (після доопрацювання), оскільки не треба буде пояснювати, що система контролю оригінальності тексту (т.зв. антиплагіату) знайшла попередню версію саме мого тексту, і що це не було його офіційним виданням.

Дякую за розуміння.

3.2 Форма та структура звіту

Вимоги до форми – мінімальні:

1. Прізвище та ім'я студента, номер групи, номер лабораторної роботи – у верньому колонтитулі. Нуменація сторінок – у нижньому колонтитулі. Титульний аркуш не потрібен.

2. На першому аркуші, угорі, – назва (тема) лабораторної роботи.

3. Далі, на першому ж аркуші та наступних – змістовна частина

Структура змістовної частини звіту:

1. Завдання саме Вашого (автора звіту) варіанту. Повне, включно з вимогами до арифметики (п'ять операцій, правоасоціативність піднесення до степеня) і т. і.

2. Граматика мови.

3. Специфікація усіх конструкцій, на які розширюється транслятор.
4. Про лексичний аналізатор коротко.
5. Про синтаксичний аналізатор коротко.
6. Формат таблиць: символів, ідентифікаторів, констант, міток (якщо використовуються).
7. Базовий приклад програми вхідною мовою для тестування транслятора.
8. Опис програмної реалізації транслятора, незмінні з минулої роботи частини описувати не треба.
9. План тестування, напр. як у Табл. 7.
10. Протокол тестування. Можна у формі скриншотів, текстових копій термінала тощо: фрагмент програми + результат обробки + ваш коментар та/чи оцінка результату
11. Висновки. Тут очікується власні оцінка/констатація/враження/зауваження автора звіту - виконавця лабораторної роботи.

План тестування може бути представлений у формі таблиці:

№	Тип випробування	Очікуванний результат	Кількість випробувань
1	базовий приклад	успішне виконання, ПОЛІЗ	1
2	вкладені конструкції	успішне виконання, ПОЛІЗ	5
3	`if` з варіантами `BoolExpr`	...	...
4	`if` з варіантами `ThenPart`	...	...
5	...	...	

Табл. 7: План тестування

Література

[1] Ахо, А. Компиляторы: принципы, технологии и инструментарий, 2-е изд. : Пер. с англ. / Альфред Ахо, Моника Лам, Рави Сети, Джефри Д. Ульман. – М. : ООО „И.Д. Вильямс”, 2008. – 1184 с.

[2] Медведева В.М. Транслятори: внутрішнє подання програм та інтерпретація [Текст]: навч.посіб. [Текст]: навч.посіб. / В.М. Медведєва, В.А. Третяк/-К.: НТУУ «КПІ»,2015.-148с.

[3] Матеріали до виконання лабораторної роботи № 3 ”Трансляція у ПОЛІЗ арифметичних виразів та інтерпретація постфіксного коду”