

ВСТУП

В даний час штучні мови, що використовують для опису предметної області текстове представлення, широко застосовуються не тільки в програмуванні, але і в інших областях. З їх допомогою описується структура всіляких документів, тривимірних віртуальних світів, графічних інтерфейсів користувача та багатьох інших об'єктів, використовуваних в моделях і в реальному світі. Для того, щоб ці текстові описи були коректно складені, а потім правильно розпізнані й інтерпретовані, використовуються спеціальні методи їх аналізу і перетворення. В основі методів лежить теорія мов і формальних граматики, а також теорія автоматів. Програмні системи, призначені для аналізу та інтерпретації текстів, називаються трансляторами.

Незважаючи на те, що до теперішнього часу розроблені тисячі різних мов і їх трансляторів, процес створення нових додатків в цій сфері не припиняється. Це зв'язано як з розвитком технології виробництва обчислювальних систем, так і з необхідністю вирішення все більш складних прикладних задач. Крім того, елементи теорії мов і формальних граматики застосовні і в інших різноманітних областях, наприклад, при описі структур даних, файлів, зображень, представлених не в текстовому, а двійковому форматі. Ці методи корисні і при розробці своїх трансляторів навіть там, де вже є відповідні аналоги. Така розробка може бути обумовлена різними причинами, зокрема, функціональними обмеженнями, відсутністю локалізації, низькою ефективністю. Тому, основи теорії мов і формальних граматики, а також практичні методи розробки трансляторів лежать у фундаменті інженерної освіти з інформатики та обчислювальної техніки.

					УКР.НТУУ"КПІ" _ТЕФ_АПЕПС_ТВ1184_13К 81-1	Лист
						5
Изм	Лист	№ докум	Подп	Дата		

1. ПОСТАНОВКА ЗАДАЧІ

Розробити транслятор з мови, що містить оператори вводу-виводу, присвоєння, а також умовного переходу й циклу наступного виду:

if < лог. Вираз > **then**

< список операторів >

else

< список операторів >

endif

for < ид. > = < вираз > **step** < лог. вираз > **to** < лог. вираз >

< список операторів >

next

Арифметичний вираз містить цілочисельні константи, імена змінних, дужки (), операції + - * / % ^ root та унарний - .

Алгоритм лексичного аналізу: розбір до роздільника.

Алгоритм синтаксичного аналізу: синтаксичний розбір методом магазинного автомату.

					УКР.НТУУ"КПІ" ТЕФ_АПЕПС ТВ1184_13К 81-1	Лист
						6
Изм	Лист	№ докум	Подп	Дата		

2. СТРУКТУРА ТРАНСЛЯТОРА

Розроблюваний транслятор складається з наступних блоків:

- 1) текстовий парсер;
- 2) лексичний аналізатор;
- 3) синтаксичний аналізатор;
- 4) генератор коду ПОЛІЗ.
- 5) виконувач ПОЛІЗу

На вхід лексичного аналізатора надходить вхідний текст програми на мові високого рівня. Текст програми складається з послідовності символів.

Метою текстового парсера є поділ вхідного тексту програми на окремі рядки, зважаючи на поточні розділювачі в граматиці.

Метою лексичного аналізатора є обробка списку рядків, представлених текстовим парсером і генерація з них лексем (об'єктів класу *Lexem*), що будуть використовуватись у подальшій обробці та генерації коду. Лексеми – окремі об'єкти, які несуть деяке значення для мови програмування, наприклад, службові слова, константи, ідентифікатори, розділові знаки, арифметичні та логічні операції та інші символи, що використовуються в даній мові. В результаті обробки лексичним аналізатором на виході одержуємо текст програми у вигляді послідовності лексем, а також допоміжних таблиць (ідентифікаторів та констант).

Інші блоки транслятора використовують послідовність лексем, складену лексичним аналізатором.

Мета синтаксичного аналізатора – це перевірка програми на правильність, відповідність граматиці мови програмування. Синтаксичний

					УКР.НТУУ"КПІ" _ТЕФ_АПЕПС_ТВ1184_13К 81-1	Лист
						7
Изм	Лист	№ докум	Подп	Дата		

аналізатор перевіряє правильну послідовність лексем, яка задана граматиною мови.

Блок генератора коду використовується для підготовки коду до його інтерпритації (виконання). Польський інверсний запис може бути результатом генерації коду.

Даний транслятор має 4 проходи: перший прохід – це розбір текстова перед-обробка, другий прохід – це лексичний аналіз, на виході якого отримуємо послідовність лексем; третій - синтаксичний аналіз для перевірки правильності послідовності лексем; четвертий - побудова польського інверсного запису. Після побудови ПОЛІЗ відбувається його виконання.

Висновки до розділу

В даній курсовій роботі розроблюваний транслятор складається з чотирьох блоків: текстового парсера, лексичного аналізатора, синтаксичного аналізатора, генератора коду ПОЛІЗ.

Текстовий ділить вхідний текст програми на окремі рядки. Лексичний аналізатор генерує з них лексеми, що будуть використовуватись у подальшій обробці та генерації коду. Даний транслятор починається саме з нього. Після лексичного аналізу виконується синтаксичний. Синтаксичний аналізатор перевіряє правильну послідовність лексем, яка задана граматиною мови. Блок генератора коду використовується для переведення послідовності операторів мови високого рівня на мову програмування низького рівня. Після побудови ПОЛІЗ відбувається його виконання, тобто виконання усієї програми.

					УКР.НТУУ"КПІ" _ТЕФ_АПЕПС_ТВ1184_13К 81-1	Лист
						8
Изм	Лист	№ докум	Подп	Дата		

3. ОПИС МЕТОДУ РОЗВ'ЯЗАННЯ ПОСТАВЛЕНОЇ ЗАДАЧІ

3.1. Побудова граматики

ГраMATика мови програмування – це формальний опис її синтаксису або форми, в яких записані окремі речення програми.

ГраMATика мови програмування розроблюваного транслятора

Таблиця 3.1. ГраMATика мови

```

<app>::=      @interface <app name> ␣
               <list of definition> @implementation ␣
               <list of operators> @end

<list of
definitions>::= <definition> | <definition>␣ <list of definitions>

<definition>::= <type> <list of var>
<type>::=      int
<list of var>::= <var> | <var>, <list of var>
<list of
operators>::=   <operator> ␣ | <operator> ␣ <list of operators>

<action>::=     <operator> ␣ | { ␣ <list of operators> ␣ }
<operator>::=   <setter> | <input> | <output> | <condition> | <cycle>
<setter>::=     <var> = <expression>
<input>::=      input(<list of var>)
<output>::=     output(<list of var>)
<cycle>::=      for <var> = <expression> step <expression> to <expression> ␣
               <action> ␣
               next

<condition>::=  if <logical expression> then ␣
               <action> ␣
               else ␣
               <action> ␣
               endif

<logical
expression>::=  <log.exp.lev1> | <log.exp.lev1> || <logical expression>
<log.exp.lev1>::<log.exp.lev2> | <log.exp.lev2> && <log.exp.lev1>
=
<log.exp.lev2>::<relation> | !<log.exp.lev2> | (<logical expression>)

```

					УКР.НТУУ"КПІ" _ТЕФ_АПЕПС_ТВ1184_13К 81-1	Лист
						9
Изм	Лист	№ докум	Подп	Дата		

```

=
<relation>::= <expression> <connotation> <expression>
<connotation>:: >= | > | <= | < | == | !=
=
<expression>::= <term> | -<term> | <term> + <expression> | <term> - <expression>
<term>::= <multiplier> | <multiplier> * <term> |
          <multiplier> / <term> | <multiplier> % <term>
<multiplier>::= <expr.response> | <multiplier> ^ <expression> |
               <multiplier> root <expression>
<expr.response>::= <var> | <const> | (<expression>)
::=
<var>::= <letter> | <var><letter> | <var><number>
<const>::= <number> | <number><const>
<letter>::= A | B | .. | Z | a | b | .. | z
<number>::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

```

3.2. Текстовий парсер

Основною функцією текстового парсера є виокремлення з вхідного тексту програму окремі рядки, розбиті роздільниками.

Алгоритм викоремлення:

- 1) у вхідному тексті програми перед і після кожного розділювача вставити додатковий пробіл (' ');
- 2) пройтись по всьому тексту програми і видалити зайві пробіли;
- 3) об'єднати входження символів '>', '<', '!', що знаходяться перед символом '='.

Список розділювачів, за якими текстовий парсер виокремлює рядки:

"\n", '+', '-', '/', '*', '^', '=', '>', ',', '<', '!', '{', '}', '(', ')', '[', ']', ' ', '&', '|', '!', ',', '%', ""

Також текстовий парсер виконує наступні функції:

- 1) видаляти зайві роздільники;
- 2) видаляти коментарі.

					УКР.НТУУ"КПІ"_ТЕФ_АПЕПС_ТВ1184_13К 81-1	Лист
						10
Изм	Лист	№ докум	Подп	Дата		

3.3. Лексичний аналіз

Метод розбору до роздільника

Як вже було вказано, лексичний розбір використовується для розбивки тексту програми на окремі лексеми (ідентифікатори, константи). Для зберігання інформації про них використовуються таблиці [1].

Процес побудови лексем виглядає наступним чином. Існуючий рядок розділяємо на лексеми символом-роздільником (пробіл). Таблиця лексем (таблиця 3.2) наведена нижче.

Коли зчитується наступний символ вхідного рядка, визначаємо, є він роздільником чи ні. Між двома роздільниками знаходиться лексична одиниця.

Спочатку лексична одиниця порівнюється з лексемами типу термінал. При збігу він записується у вихідну таблицю лексем з відповідним кодом. В іншому випадку елемент класифікується як можливий ідентифікатор.

Якщо елемент класифікується як можливий ідентифікатор, то перевіряється таблиця ідентифікаторів. Якщо такого символу ще не має, він заноситься до таблиці ідентифікаторів. Якщо є, символ заноситься лише у вихідну таблицю лексем. Аналогічно аналізатор визначає константи.

					УКР.НТУУ"КПІ" _ТЕФ_АПЕПС_ТВ1184_13К 81-1	Лист
						11
Изм	Лист	№ докум	Подп	Дата		

Таблиця 3.2. Таблиця лексем

@interface	1	endif	16	/	31
@implementation	2	{	17	%	32
@end	3	}	18	*	33
int	4	(19	^	34
input	5)	20	root	35
output	6	=	21	,	36
\n	7	equ	22	and	37
for	8	!=	23	or	38
to	9	>	24	[39
step	10	<	25]	40
do	11	>=	26	id	41
next	12	<=	27	const	42
if	13	!	28		
then	14	+	29		
else	15	-	30		

Таким чином на рівень лексичного аналізатора можна винести перевірку правильності ідентифікаторів та констант.

Приклад роботи лексичного аналізатора

Лексичний аналізатор перевіряє усі символи на відповідність таблиці лексем. Якщо буде знайдено помилку, користувач отримає відповідне повідомлення з вказаним рядком у вихідному тексті.

Основні 2 помилки, які оброблює лексичний аналізатор:

- 1) оголошення змінної, яка вже була оголошена (рисунок 3.3);
- 2) використання неоголошеної змінної (рисунок 3.4).

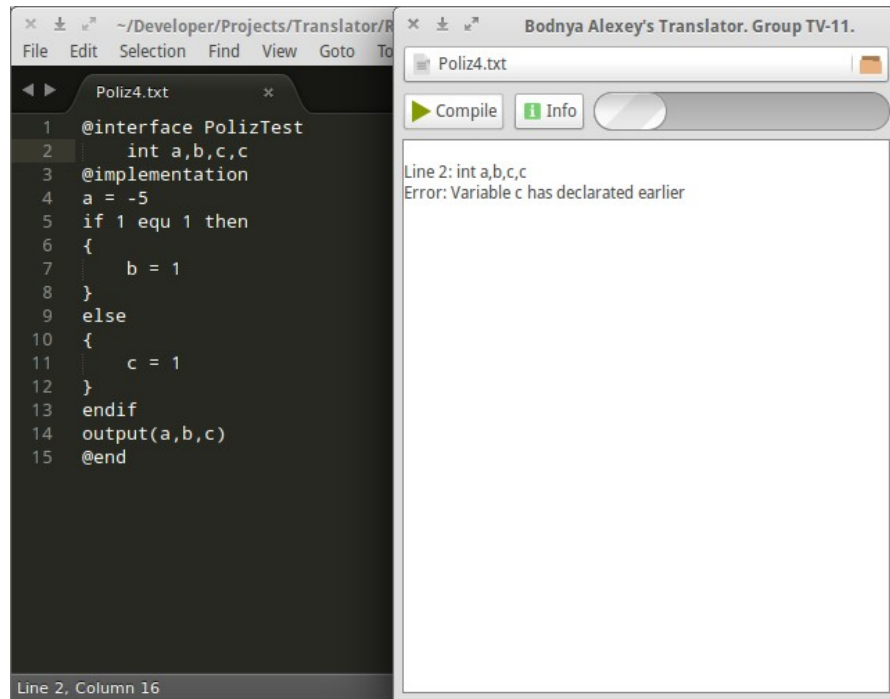


Рисунок 3.3 – Приклад програми з помилкою: оголошення змінної, яка вже була оголошена

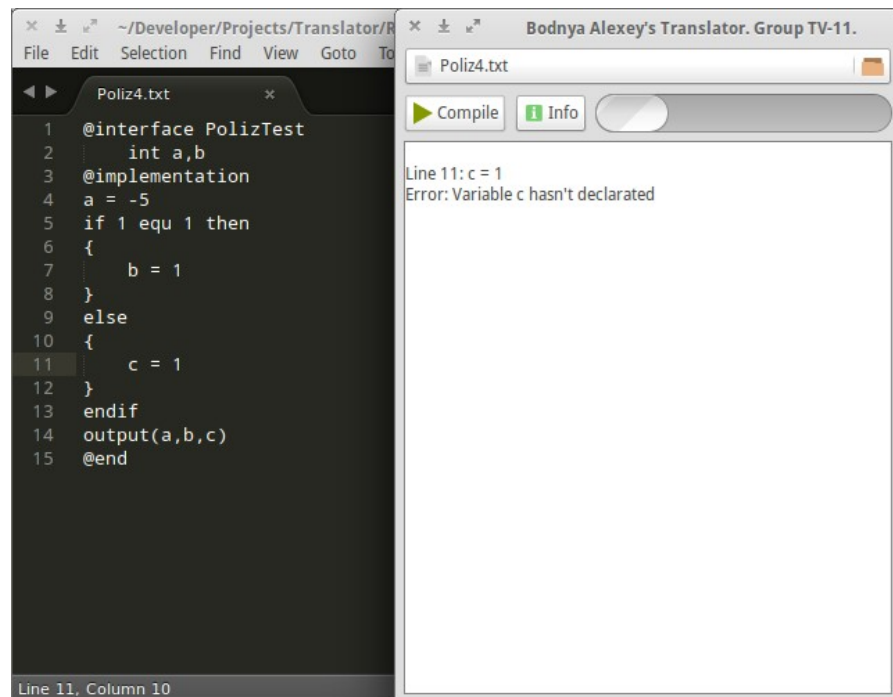


Рисунок 3.4 – Приклад програми з помилкою: використання неоголошеної змінної

					УКР.НТУУ"КПІ" _ТЕФ_АПЕПС_ТВ1184_13К 81-1	Лист
						13
Изм	Лист	№ докум	Подп	Дата		

3.4. Синтаксичний аналіз

Синтаксичний аналіз методом магазинного автомату

Після розбору на лексеми, в трансляторі йде фаза синтаксичного аналізу, ціль якого – перевірка правильності порядку лексем, інакше кажучи, перевірка правильності початкового ланцюжка [2].

Реалізація синтаксичного аналізатора методом магазинного автомата базується у відповідності з шаблоном проектування State Machine (укр. *Скінчений Автомат*).

Скінченний автомат, є особливим видом автомату — абстракції, що використовується для описання шляху зміни стану об'єкта в залежності від досягнутого стану та інформації отриманої ззовні. Його особливістю є скінченність множини станів автомату. Поняття скінченного автомата було запропоновано в якості математичної моделі технічних приладів дискретної дії, оскільки будь який такий пристрій (в силу скінченності своїх розмірів) може мати тільки скінченну кількість станів.

При реалізації магазинного автомату використовується таблиця станів, яка перевіряє правильне розташування лексем, відповідно заданої граматики.

					УКР.НТУУ"КПІ"_ТЕФ_АПЕПС_ТВ1184_13К 81-1	Лист
						14
Изм	Лист	№ докум	Подп	Дата		

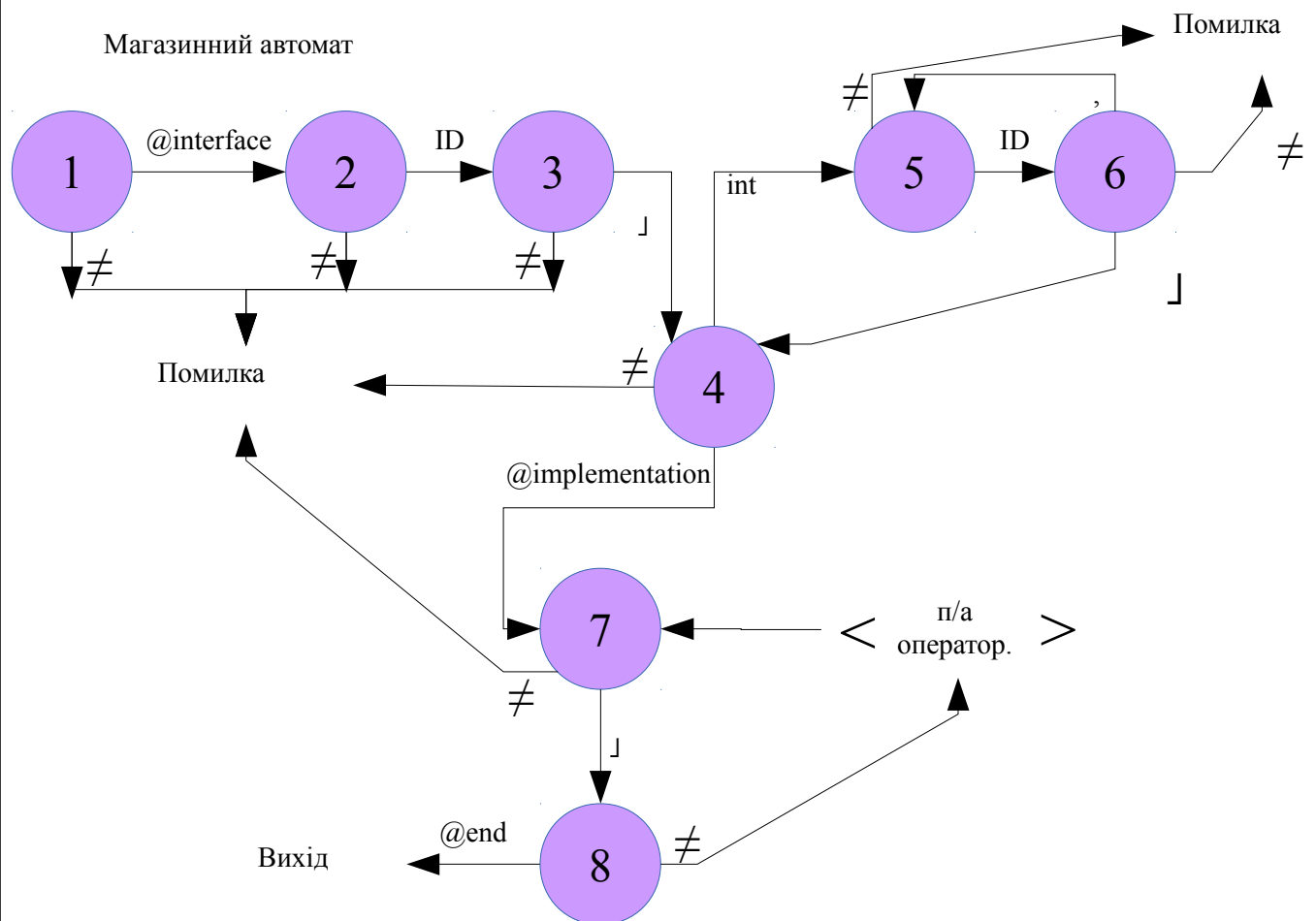
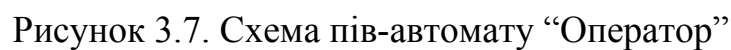


Рисунок 3.5. Схема магазинного автомату

Таблиця 3.6. Таблиця магазинного автомату

α	Мітка переходу	β	Стек	Семантична підпрограма
1	@implementation	2		[≠] помилка
2	ID	3		[≠] помилка
3	ENTER	4		[≠] помилка
4	Int	5		[≠] помилка
	@implementation	7		
5	ID	6		[≠] помилка
6	ENTER	4		[≠] помилка
	,	5		
7	ENTER	8		[≠] помилка
8	@end			[=] вихід [≠] п/а оператор ↓7



α	Мітка переходу	β	Стек	Семантична підпрограма
11	ID	12		[\neq] помилка
	input	14		
	output	17		
	if	п/а лог.вираз	↓20	
	for	27		
12	=	п/а вираз	↓13	[\neq] помилка
13	¢			[\neq] вихід
14	(15		[\neq] помилка
15	ID	16		[\neq] помилка
16)			[=] вихід
	,	15		[\neq] помилка
17	(18		[\neq] помилка
18	ID	19		[\neq] помилка
	CONST	19		
19	,	18		[\neq] помилка
)			[=] вихід
20	then	21		[\neq] помилка
21	ENTER	п/а сп.опер.	↓22	[\neq] помилка
22	ENTER	23		[\neq] помилка
23	else	24		[\neq] помилка
	endif			[=] вихід
24	ENTER	п/а сп.опер.	↓25	[\neq] помилка
25	ENTER	26		[\neq] помилка
26	endif			[\neq] помилка [=] вихід
27	ID	28		[\neq] помилка
28	=	п/а вираз	↓29	[\neq] помилка
29	step	п/а вираз	↓30	[\neq] помилка
30	to	п/а вираз	↓31	[\neq] помилка
31	do	32		[\neq] помилка
32	ENTER	п/а сп.опер.	↓33	[\neq] помилка
33	ENTER	34		[\neq] помилка
34	next			[\neq] помилка [=] вихід

Таблиця 3.8. Таблиця пів-автомату “Оператор”

					УКР.НТУУ”КПІ”_ТЕФ_АПЕПС_ТВ1184_13К 81-1	Лист
						17
Изм	Лист	№ докум	Подп	Дата		

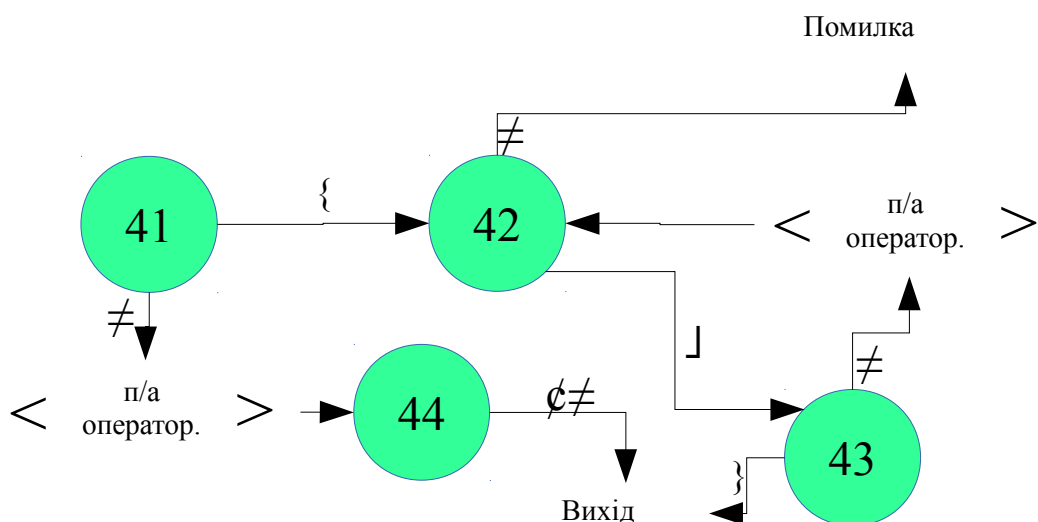


Рисунок 3.9. Схема пів-автомату “Блок операторів”

Таблиця 3.10. Таблиця пів-автомату “Блок операторів”

α	Мітка переходу	β	Стек	Семантична підпрограма
41	{	42		[≠] п/а оператор ↓44
42	ENTER	43		[≠] помилка
43	}			[=] вихід [≠] п/а оператор ↓42
44	⌋			[≠] вихід

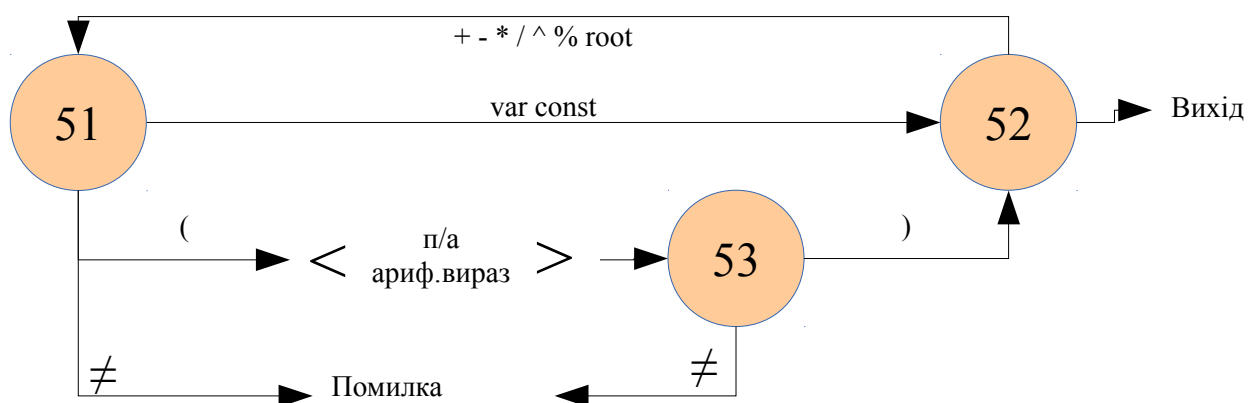


Рисунок 3.11. Схема пів-автомату “Арифметичний вираз”

Таблиця 3.12. Таблиця пів-автомату “Арифметичний вираз”

α	Мітка переходу	β	Стек	Семантична підпрограма
51	ID	52		[≠] помилка
	CONST	52		

	(п/а вираз	↓53	
52	+	51		[≠] вихід
	-	51		
	*	51		
	/	51		
	^	51		
	%	51		
	root	51		
53)	52		[≠] помилка

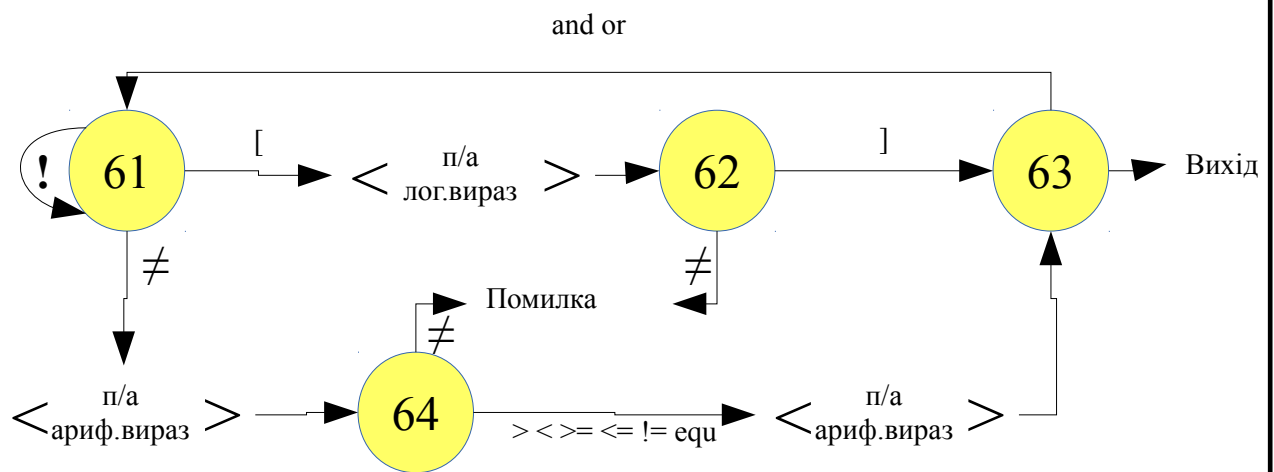


Рисунок 3.13. Схема пів-автомату “Логічний вираз”

Таблиця 3.14. Таблиця пів-автомату “Логічний вираз”

α	Мітка переходу	β	Стек	Семантична підпрограма
61	!	61		[≠] п/а вираз ↓64
	[п/а лог. вираз	↓62	
62]	63		[≠] помилка
63	and	61		[≠] вихід
	or	61		
64	>	п/а вираз	↓63	[≠] помилка
	>=	п/а вираз	↓63	
	<	п/а вираз	↓63	
	<=	п/а вираз	↓63	
	!=	п/а вираз	↓63	
	equ	п/а вираз	↓63	

Результат роботи синтаксичного аналізатора

Якщо тестова програма написана правильно (без помилок), буде отримано повідомлення про успішне завершення синтаксичного аналізатора (рисунок 3.15.), в іншому випадку – повідомлення про помилку із зазначенням причини та номера рядку, в якому вона виникла (рисунок 3.16.).

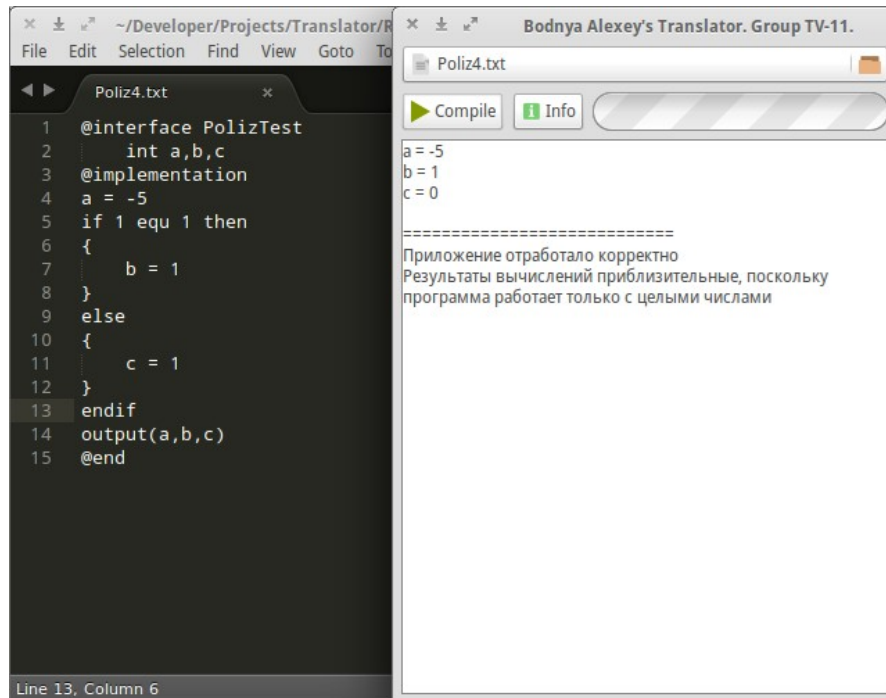


Рисунок 3.15. – Приклад програми без синтаксичних помилок

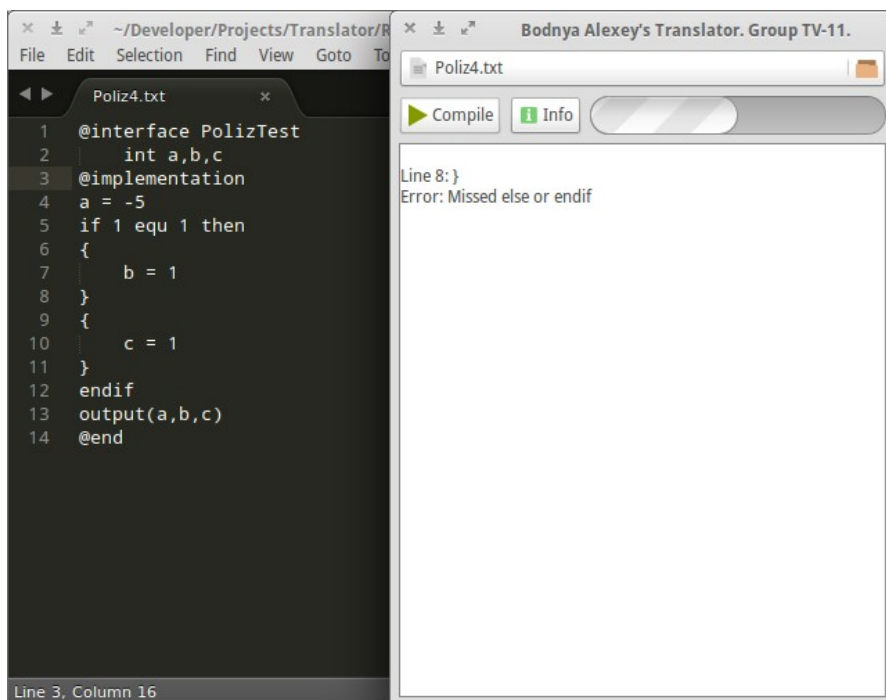


Рисунок 3.16. – Приклад програми з синтаксичною помилкою

					УКР.НТУУ"КПІ" ТЕФ_АПЕПС ТВ1184_13К 81-1	Лист
						20
Изм	Лист	№ докум	Подп	Дата		

3.5. Побудова ПОЛІЗ

Вихідний код програми є досить складним для виконання, тому для виконання програм потрібно використовувати проміжні форми. При написанні даного транслятора проміжною формою є польський інверсний запис (ПОЛІЗ).

Правила складання:

- 1) ідентифікатори в ПОЛІЗ ідуть в тому ж порядку, що і в початковому (інфіксному) записі;
- 2) операції в ПОЛІЗ, ідуть у тому порядку, в якому вони повинні обчислюватись при перегляді зліва направо (порядок може не збігатися з початковим);
- 3) операції в ПОЛІЗ, розташовані безпосередньо за своїми операндами.

При побудові ПОЛІЗ використовуємо алгоритм Дейкстри [2], що базується на явному завданні пріоритетів операцій. Пріоритети операцій розставляються таким чином, щоб пріоритет операції, яка виконується раніше, був більший пріоритету операції, яка виконується пізніше (таблиця 3.17.).

Таблиця 3.17. Пріоритети операцій

Операція	Пріоритет
([for if input output	0
)] to next then else do step	1
=	2
or	3
and	4
!	5

> >= < <= equ !=	6
+ -	7
* / %	8
^ root	9

Алгоритм побудови ПОЛІЗ зводиться до наступного:

- 1) ідентифікатори та константи проходять від входу прямо в ПОЛІЗ, операції потрапляють на вихід через стек;
- 2) якщо пріоритет операції у стеку, більше або дорівнює пріоритету поточної операції, то операція із стека передається на вихід і п.2 повторюється, інакше поточна операція заноситься в стек;
- 3) якщо стек порожній, поточна операція заноситься в стек;
- 4) якщо вхідний ланцюжок порожній, то всі операції із стека виштовхуються на вихід ознакою кінця рядка ENTER.

ПОЛІЗ операторів мови

Оператор вводу: **input** (a_1, a_2, \dots, a_n) трансліюється в ПОЛІЗ наступним чином: $a_1 a_2 \dots a_n$ **input**.

()- відкриваюча та закриваюча дужки. Пріоритети відповідно 0 і 1. Працюють по своєму пріоритету нічого не заносючи.
, ігнорується.

Оператор виводу: **output** (a_1, a_2, \dots, a_n) трансліюється в ПОЛІЗ аналогічно оператору вводу.

Для побудови ПОЛІЗ зручно використовувати команди умовного та безумовного переходу з відповідними операндами:

m БП – безумовний перехід до мітки **m**;

<лог.врж> **m** УПХ – перехід на мітку **m**, якщо вираз хибний.

Опертор умовного переходу: **if** <лог вираз> **then ENTER** <опер-и 1> **else** <опер-и 2> **endif**

					УКР.НТУУ"КПІ"_ТЕФ_АПЕПС_ТВ1184_13К 81-1	Лист
						22
Изм	Лист	№ докум	Подп	Дата		

Для полегшення побудови ПОЛІЗ розглянемо блок-схему (рисунок 3.18).

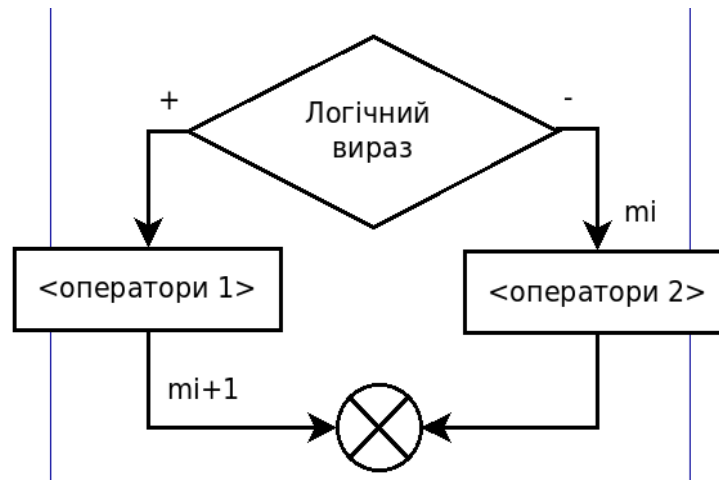


Рисунок 3.18. – Блок-схема оператора умовного переходу

Для даного умовного оператора ПОЛІЗ має наступний вигляд:

<лог вираз> m_i УПЛ <опер-и 1> m_{i+1} БП m_i <опер-и 2> m_{i+1}

if як і відкриваюча дужка із пріоритетом 0 записується в стек та використовується як «зберігач» робочих міток.

then із пріоритетом 1 генерує робочу мітку m_i , виштовхує зі стека всі символи до **if** і записує їх у ПОЛІЗ, а також записує в ПОЛІЗ m_i УПЛ.

else із пріоритетом 1 генерує робочу мітку m_{i+1} , виштовхує зі стека всі символи до **if** і записує їх у ПОЛІЗ, генерує на вихід m_{i+1} П, а також ініціалізує мітку m_i .

endif із пріоритетом виштовхує зі стека всі символи до **if** (його видаляє) і записує їх у ПОЛІЗ, а також ініціалізує мітку m_{i+1} .

Приведемо приклад побудови ПОЛІЗа оператора умовного оператора (таблицям 3.19.).

УКР.НТУУ"КПІ" ТЕФ_АПЕПС ТВ1184_13К 81-1					Лист
					23
Изм	Лист	№ докум	Подп	Дата	

Поліз		a		b	> <i>m1</i> УПЛ	a		5	= <i>m2</i> БП <i>m1</i> :	a		0	= <i>m2</i> :
Стек	if	if	> if	> if	if <i>m1</i>	if <i>m1</i>	= if <i>m1</i>	= if <i>m1</i>	if <i>m1</i> , <i>m2</i>	if <i>m1</i> , <i>m2</i>	= if <i>m1</i> , <i>m2</i>	= if <i>m1</i> , <i>m2</i>	
Вхідний код	if	a	>	b	then <i>m1</i>	a	=	5	else <i>m2</i>	a	=	0	endif

Таблиця 3.19. Побудова ПОЛІЗа умовного оператора

Вихідний поліз:

a b > *m1* УПЛ a 5 := *m2* БП *m1* : a 0 := *m2* :
1 2 3 4 5 6 7 8 9 10 11 12 13 14

Оператор циклу: **for** iter = <start> **step** <step> **to** <end> **ENTER**

<operators> **next**

Для полегшення побудови ПОЛІЗ будується блок-схема (рисунок 3.20.).

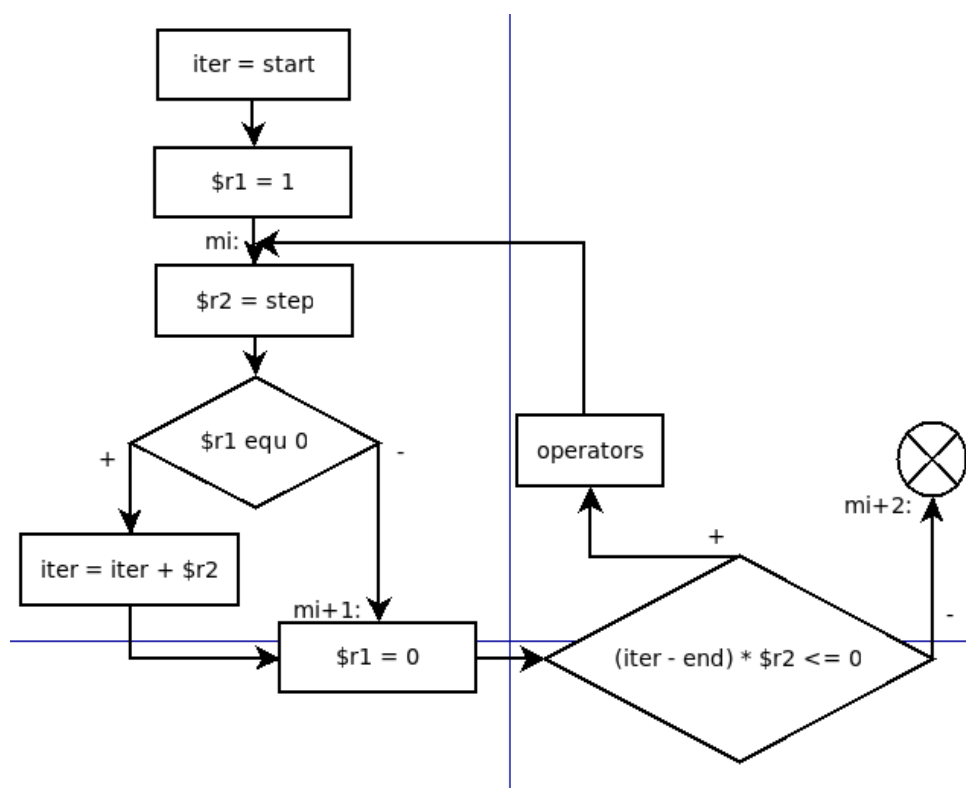


Рисунок 3.20. – Блок-схема оператора циклу

де *iter* – параметр циклу – змінна (ідентифікатор);
 <start> – початкове значення параметра циклу – вираз;
 <end> – кінцеве значення параметра циклу – вираз;
 <step> – вираз значення кроку циклу.

ПОЛІЗ для оператора циклу має наступний вигляд:

iter start = \$r_1 1 = m_i: \$r_2 step = \$r_1 0 equ m_{i+1} **УПЛ** *i i \$r_2 + = m_{i+1}: \$r_1 0 =*
*iter end - \$r_2 * 0 <= m_{i+2}* **УПЛ** *operators m_i БП m_{i+1}:*

for - грає роль відкриваючої дужки. Має пріоритет 0. Генерує 3 робочі мітки. У стек записується *for* з мітками "*for m_i m_{i+1} m_{i+2}*". В поліз нічого не пишеться .

Символ "=" - діє згідно свого пріоритету.

step - є закриваючою дужкою для попереднього арифметичного виразу. Має пріоритет 1. Виштовхує з стека все, до *for* виключно. Генерує додаткові регістри *r_j* , *r_j + 1* . Записує в полізім *r_j 1 = m_i: r_{j+1}* ;

to - має пріоритет 1. Виштовхує з стека все до *for* виключно. В поліз генерує запис *= r_j 0 m_{i+1} УПЛ iter iter r_{j+1} + = m_{i+1}: r_j 0 := iter* ;

do - є закриває дужкою для попереднього арифметичного виразу. Має пріоритет 1. Виштовхує зі стека все до *for* виключно. Генерує в поліз - *r_{i+1} * 0 <= m_{i+2} УПЛ* . Лічильник додаткових регістрів -2 (звільняються регістри *r_j* , *r_j + 1*) ;

endif - очищає стек до *for*. На вихід генерує *m_i БП m_{i+2}*: і видаляє запис *for* з стека.

Наведемо приклад побудови ПОЛІЗа оператора циклу (таблиця 3.21.).

Вхідний код:

for iter := 1 step 2 to 7 do a := a + 1 endif

Вихідний поліз:

iter 1 := r1 1:= m1: r2 2 := r1 0 = m2 УПЛ iter iter r2 + := m2: r1 0 := iter
*7 - r2 * 0 <= m3 УПЛ a a 1 + := m1 БП m3:*

					УКР.НТУУ"КПІ" _ТЕФ_АПЕПС_ТВ1184_13К 81-1	Лист
						25
Изм	Лист	№ докум	Подп	Дата		

п о л і з		iter		1	= r1 1 = m1: r2	2	= r1 0 equ m2 УПЛ iter iter r2 + = m2: r1 0 = iter	7	- r2 * 0 <= m3 УПЛ	a		a		1	= m1 БП m3:
с т е к	for m1 m2 m3	for m1 m2 m3	= for m1 m2 m3	= for m1 m2 m3	for m1 m2 m3	for m1 m2 m3	for m1 m2 m3	for m1 m2 m3	for m1 m2 m3	for m1 m2 m3	= for m1 m2 m3	= for m1 m2 m3	+ = for m1 m2 m3	+ = for m1 m2 m3	
В х ц е п	for m1 m2 m3	iter	=	1	step r1, r2	2	to	7	do r - 2	a	=	a	+	1	endif

3.21. Таблица побудови ПОЛІЗу оператору циклу.

Приклад побудови ПОЛІЗ

Побудований ПОЛІЗ можна переглянути у вікні браузера, натиснувши кнопку “Info” (рисунок 3.22.).

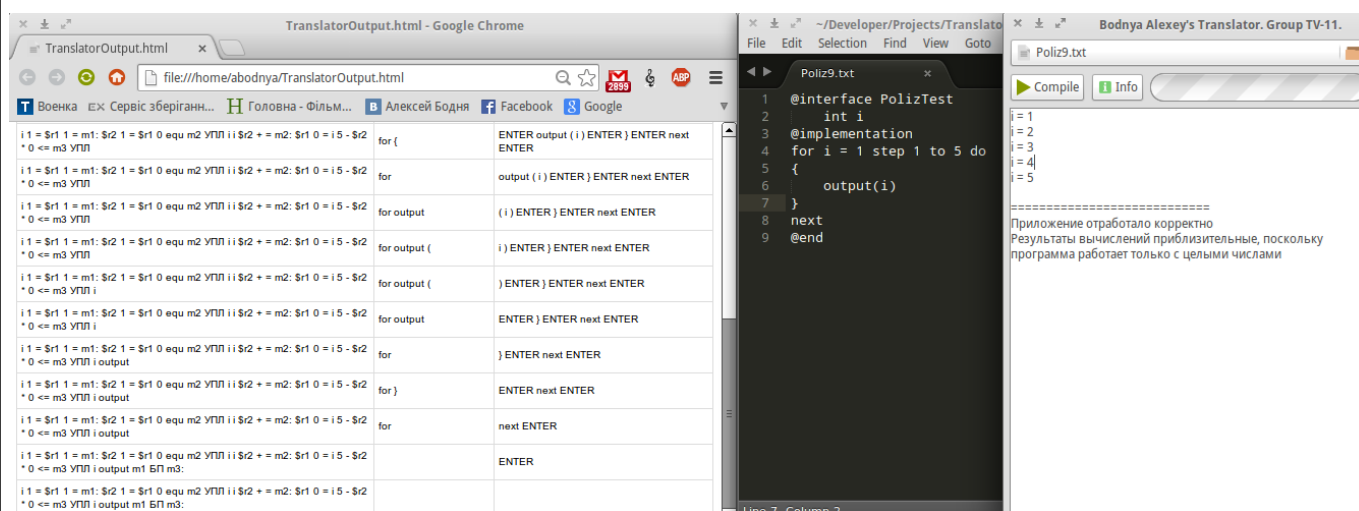


Рисунок 3.22. – Побудований ПОЛІЗ

					УКР.НТУУ”КПІ”_ТЕФ_АПЕПС_ТВ1184_13К 81-1	Лист
						26
Изм	Лист	№ докум	Подп	Дата		

3.5. Виконання ПОЛІЗ

Алгоритм виконання ПОЛІЗ

Для коректного пересування по окремим операторам, був введений вказівник на початок поточного оператора — *commIter*.

Для пересування по масиву лексем полізу використовується ітератор *i*.

Перед початком компіляції, вказівник *commIter* встановлюється в 0.

- 1) Якщо поточна лексема — “=”, тоді:
 - a) лексема за адресою *commIter* вважається ідентифікатором;
 - b) набір лексем, починаючи з *commIter+1* і закінчуючи попередньою лексемою (*i-1*) вважається арифметичним виразом;
 - c) результат підрахунку виразу записується як значення змінної з ідентифікатором, який знаходиться за адресою *commIter*;
 - d) *commIter* встановлюється в *i+1*.
- 2) Якщо поточна лексема — “output”, тоді:
 - a) лексеми на відріжку від *commIter* до *i-1* вважаються ідентифікаторами або константами і виводяться у форматі “назва змінної = значення змінної”;
 - b) *commIter* встановлюється в *i+1*.
- 3) Якщо поточна лексема — “input”, тоді:
 - a) лексеми на відріжку від *commIter* до *i-1* вважаються ідентифікаторами і по черзі вводяться в спеціальне поле вводу;
 - b) *commIter* встановлюється в *i+1*.
- 4) Якщо поточна лексема — УПЛ, тоді:
 - a) лексема за адресою *i-1* вважається відкриваючою міткою (без “двокрапки”);

					УКР.НТУУ”КПІ”_ТЕФ_АПЕПС_ТВ1184_13К 81-1	Лист
						27
Изм	Лист	№ докум	Подп	Дата		

- b) лексеми на відрізку від *commIter* до *i-2* вважаються логічним виразом;
 - c) якщо підрахований вираз *неправда (false)* — відбувається перехід на знайдену мітку, тобто в *i* записується значення адреси відповідної закриваючої мітки (з “двокрапкою”);
 - d) *commIter* встановлюється в *i+1*.
- 5) Якщо поточна лексема — відкриваюча мітка, а наступна — БП, тоді:
- a) попередня умова була виконана, і компіляція дійшла до невикористаного “else” блоку;
 - b) відбувається перехід на закриваючу мітку, що відповідає поточній лексемі (відкриваючі мітці);
 - c) *commIter* встановлюється в *i+1*.
- 6) Якщо поточна лексема — закриваюча мітка, тоді:
- a) був виконаний “else” блок, наразі ця мітка не потрібна, поточна лексема пропускається;
 - b) *commIter* встановлюється в *i+1*.
- 7) В інших випадках відбувається перехід до наступної лексеми.

Особливості алгоритму виконання ПОЛІЗ

Математичні операції $+, -, *, /, \text{root}, \%, ^$ є бінарними операціями та застосовуються для двох верхніх комірок стеку. Результат операції заноситься в перший операнд. Операндами можуть бути константи та ідентифікатори.

Операції порівняння $>, >=, <, <=, \text{equ}, !=$ є бінарними операціями та застосовуються до двох верхніх комірок стека. Результат операції заноситься в перший операнд. В якості результату може бути 1 (true, істина) або 0 (false, неправда). Операндами можуть бути константи та ідентифікатори.

					УКР.НТУУ”КПІ”_ТЕФ_АПЕПС_ТВ1184_13К 81-1	Лист
						28
Изм	Лист	№ докум	Подп	Дата		

Операція заперечення ! є унарною операцією. Застосовується до верхньої комірки стеку.

Логічні операції and та or є бінарними операціями. Застосовуються до двох верхніх комірок стеку. Результат операції заноситься в перший операнд.

Операція присвоєння = є бінарною операцією. Першим операндом є ідентифікатор, другим – ідентифікатор або константа. В результаті значенню першого операнда присвоюється значення другого (в таблиці ідентифікаторів).

Операція безумовного переходу БП є унарною операцією. Операндом є мітка. В результаті відбувається перехід на вказану мітку.

Операція умовного переходу УПЛ є бінарною операцією. Першим операндом є булеве значення (0 або 1), другим – мітка. Якщо значення першого операнда – фальш, то виконуємо перехід на вказану мітку.

Операція вводу є поліарною операцією. Користувачу видається повідомлення про введення даних, після чого, відбувається присвоєння.

Операція виводу є поліарною операцією. Операнди - ідентифікатори. В результаті виконання отримуємо вивід даних у текстовому полі.

					УКР.НТУУ"КПІ" _ТЕФ_АПЕПС_ТВ1184_13К 81-1	Лист
						29
Изм	Лист	№ докум	Подп	Дата		

Висновки до розділу

В даній роботі було сформовано граматику мови високого рівня, що складається з арифметичних та логічних операцій, операторів присвоєння, циклів, операторів умови, вводу та виводу даних.

На основі цієї мови складені таблиці лексем, класів символів та діаграму станів лексичного аналізатора.

Синтаксичний аналізатор, використовуючи шаблон проектування Кінцевий Автомат (State Machine), перевіряє правильність розташування лексем. Для кращого розуміння, були надані таблиці та діаграми використаного автомату,

Щоб реалізувати виконання програми використано проміжну форму подання програми – польський інверсний запис.

Для наглядного відображення роботи транслятора було надано тестові зображення програми.

					УКР.НТУУ"КПІ" _ТЕФ_АПЕПС_ТВ1184_13К 81-1	Лист
						30
Изм	Лист	№ докум	Подп	Дата		

4. МЕТОДИКА РОБОТИ КОРИСТУВАЧА З ПРОГРАМНОЮ СИСТЕМОЮ

Програмну частину транслятора було розроблено в середовищі MonoDevelop – мова програмування C#.

Розроблена програмна система є досить простою в користуванні. В головному вікні програми (рисунок 4.1) користувач здатен

- 1) відкрити файл з уже готовим текстом програми, натиснувши кнопку з зображенням теки;
- 2) виконати її трансляцію, натиснувши кнопку “Compile”;
- 3) переглянути таблицю побудову ПОЛІЗу, натиснувши “Info”;
- 4) переглянути результат роботи програми.

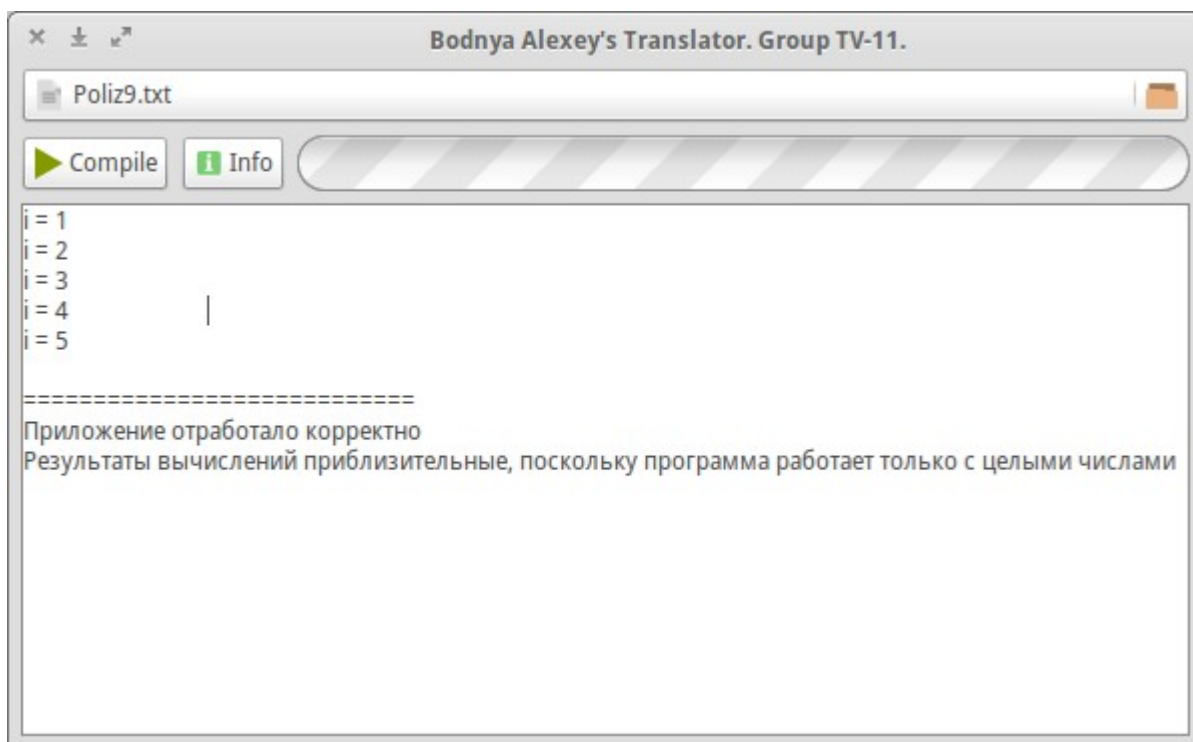
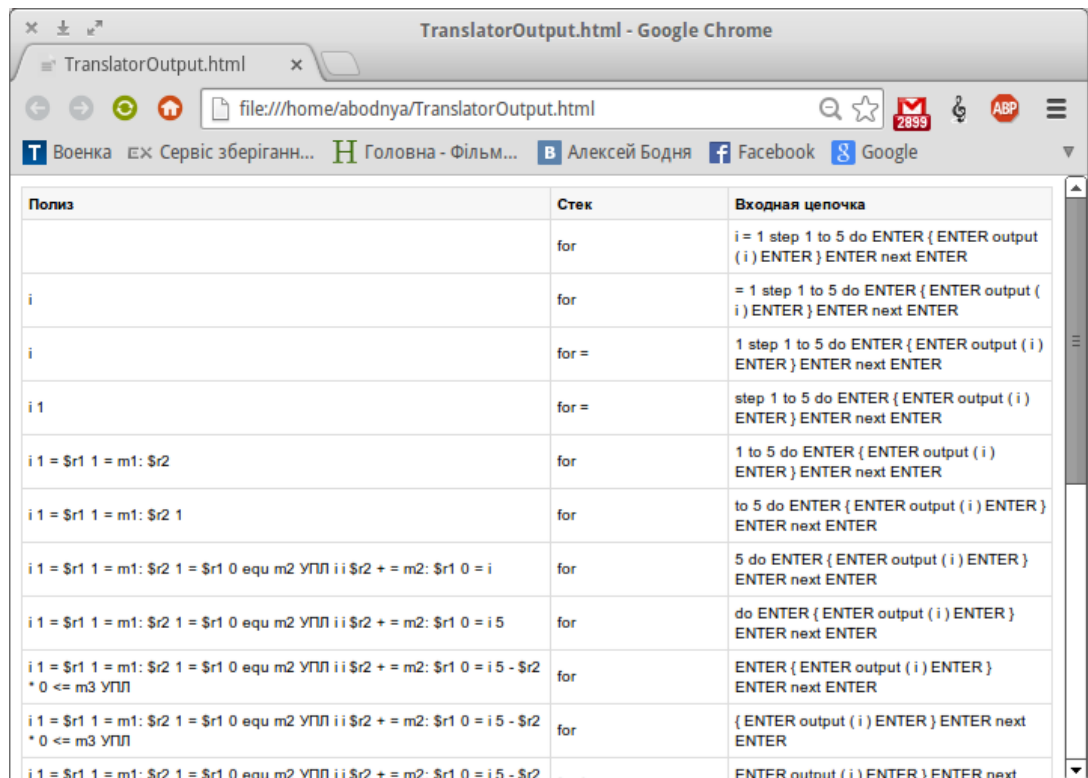


Рисунок 4.1 – Головне вікно програми

					УКР.НТУУ”КПІ”_ТЕФ_АПЕПС_ТВ1184_13К 81-1	Лист
						31
Изм	Лист	№ докум	Подп	Дата		

Кнопка «Compile» запускає на виконання лексичний та синтаксичний аналізатори, при успішному їх виконанні відбувається побудова ПОЛІЗу та його виконання (рисунок 4.1). В разі помилки, спливає повідомлення та заноситься в текстове поле, в яких вказується причина помилки та номер рядка, в якій вона була знайдена (рисунки 3.3, 3.4, 3.16).

Кнопка «Info» викликає інтернет-браузер, в якому відображається побудований ПОЛІЗ (рисунок 4.2).



Полиз	Стек	Входная цепочка
	for	i = 1 step 1 to 5 do ENTER { ENTER output (i) ENTER } ENTER next ENTER
i	for	= 1 step 1 to 5 do ENTER { ENTER output (i) ENTER } ENTER next ENTER
i	for =	1 step 1 to 5 do ENTER { ENTER output (i) ENTER } ENTER next ENTER
i 1	for =	step 1 to 5 do ENTER { ENTER output (i) ENTER } ENTER next ENTER
i 1 = \$r1 1 = m1: \$r2	for	1 to 5 do ENTER { ENTER output (i) ENTER } ENTER next ENTER
i 1 = \$r1 1 = m1: \$r2 1	for	to 5 do ENTER { ENTER output (i) ENTER } ENTER next ENTER
i 1 = \$r1 1 = m1: \$r2 1 = \$r1 0 equ m2 УПЛ ii \$r2 += m2: \$r1 0 = i	for	5 do ENTER { ENTER output (i) ENTER } ENTER next ENTER
i 1 = \$r1 1 = m1: \$r2 1 = \$r1 0 equ m2 УПЛ ii \$r2 += m2: \$r1 0 = i 5	for	do ENTER { ENTER output (i) ENTER } ENTER next ENTER
i 1 = \$r1 1 = m1: \$r2 1 = \$r1 0 equ m2 УПЛ ii \$r2 += m2: \$r1 0 = i 5 - \$r2 * 0 <= m3 УПЛ	for	ENTER { ENTER output (i) ENTER } ENTER next ENTER
i 1 = \$r1 1 = m1: \$r2 1 = \$r1 0 equ m2 УПЛ ii \$r2 += m2: \$r1 0 = i 5 - \$r2 * 0 <= m3 УПЛ	for	{ ENTER output (i) ENTER } ENTER next ENTER
i 1 = \$r1 1 = m1: \$r2 1 = \$r1 0 equ m2 УПЛ ii \$r2 += m2: \$r1 0 = i 5 - \$r2		ENTER output (i) ENTER } ENTER next

Рисунок 4.2 – Результат виконання програми
(ПОЛІЗ у вікні браузера Google Chrome)

Висновки до розділу

Розроблена програмна система є досить простою в користуванні. В головному вікні програми користувач здатен ввести відкрити файл з уже готовим текстом програми та запустити її на виконання. Також можна переглянути таблицю побудови ПОЛІЗу, натиснувши на кнопку «Info».

При натисканні на кнопку «Compile» запускається на виконання лексичний та синтаксичний аналізатори, потім будується ПОЛІЗ, який можна переглянути натиснувши кнопку «Info». Після чого програма виконується, а результати виводу програми заносяться в текстове поле програми.

					УКР.НТУУ"КПІ"_ТЕФ_АПЕПС_ТВ1184_13К 81-1	Лист
						33
Изм	Лист	№ докум	Подп	Дата		

ВИСНОВКИ

В курсовій роботі було розглянуто основні принципи розробки компілятора, розроблено власну мову програмування високого рівня. В даній мові реалізовано умовний оператор, оператор циклу, вводу та виводу числових даних, а також арифметичні та логічні операції.

В ході виконання було розроблено лексичний і синтаксичний аналізатори для перевірки коректності написаних програм. Також для полегшення виконання програми було використано проміжну форму ПОЛІЗ.

Транслятор надає можливість продемонструвати виконання програми мови високого рівня та ознайомитися з курсом дисципліни «Основи розробки трансляторів-2».

					УКР.НТУУ"КПІ"_ТЕФ_АПЕПС_ТВ1184_13К 81-1	Лист
						34
Изм	Лист	№ докум	Подп	Дата		

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Медведєва, В.М. Транслятори: лексичний та синтаксичний аналізатори. Навчальний посібник/ В.М. Медведєва, В.А. Третяк. – К : НТУУ “КПІ”, 2012.- 150с.
2. Медведєва, В.М. Основи побудови компіляторів Навчальний посібник/ В.М. Медведєва, В.Г. Сліпченко. – К : ІЗМН, 2004.- 104с.
3. Mono Documentation — Режим доступа : URL : <http://docs.gomono.com/> Название с экрана.
4. Руководство по программированию на С# — Режим доступа : URL : <http://msdn.microsoft.com/ru-ru/library/67ef8sbd.aspx> Название с экрана.
5. Справочник CSS | htmlbook.ru — Режим доступа : URL : <http://htmlbook.ru/css> Название с экрана.

					УКР.НТУУ”КПІ”_ТЕФ_АПЕПС_ТВ1184_13К 81-1	Лист
						35
Изм	Лист	№ докум	Подп	Дата		