УДК 681.3.06

В.Г. Левицький, аспір.
Житомирський інженерно-технологічний інститут

МОВА ВИЗНАЧЕННЯ ВХІДНИХ ДАНИХ ПРОГРАМНОГО СЕРЕДОВИЩА ГЕНЕРАЦІЇ МОВНИХ ПРОЦЕСОРІВ "FancyCC 3.0"

(Науковий керівник – к.т.н., доц. М.М. Колодницький)

Розглянуто спосіб визначення вхідних даних програмного середовища генерації мовних процесорів "FancyCC 3.0". Представлена мова опису формальних граматик мов програмування та проблемно-орієнтованих мов. Схематично окреслено дії транслятора мови опису під час синтаксичного розбору. Запропоновано спосіб реалізації синтаксичного підсвічування даної мови. Наведено приклад роботи користувача у середовищі, що розглядається.

Вступ

При розробці прикладних програмних комплексів часто виникає проблема реалізації зручного інтерфейсу з користувачем. Одним з шляхів вирішення цієї проблеми є використання в програмі певних проблемно-орієнтованих мов, які дозволили б користувачу формулювати задачу в простому та звичному вигляді [1–3]. Побудова трансляторів таких мов є нетривіальною проблемою у випадку ручного конструювання компілятора. Тому загальноприйнятою практикою є використання певних інструментальних засобів автоматизованої побудови трансляторів. На даний час розроблено вже досить великий арсенал подібних програм: популярний AT&T Lex і Yacc [7–10], пакети програм Cocktail [11], Gentle [12], PCCTS [13], COCOM, Eli, Amsterdam Compiler Kit [14], а також генератори лексичних і синтаксичних аналізаторів Anagram, Bison, Byacc, Coco, Cogencee, Depot4, Eag, Flex, Holub, Lisa, Llgen, Lalrgen, Mango, Muskox, Newyacc, Precc, Rdp, Scangen, Visualparse++, Yacc, Yacc++, Yooc, Zlex, Zyacc та ін. Можна однак стверджувати, що потреба у створенні нових інструментальних засобів автоматизованої побудови мовних процесорів все ще існує [4, 5]. Це, власне, підтверджується динамікою прогресу в області програмного забезпечення, що розглядається, – останні вдалі розробки середовищ генерації трансляторів датуються другою половиною дев'яностих років.

Для автоматизації побудови лінгвістичного забезпечення програмного комплексу "DSR Open Lab 1.0" [2, 6] було розроблено і використовувалось програмне середовище генерації компіляторів "FancyCC 3.0" [5]. Розглянемо далі, яким чином було вирішено проблему організації процесу визначення вхідних даних в розглядуваному програмному середовищі.

Мова опису формальних граматик

Як було відмічено вище, створення проблемно-орієнтованої мови є досить поширеним методом вирішення проблеми організації інтерфейсу з користувачем. Саме таким чином організовується введення вхідних даних і в даному випадку. Для того, щоб мати зручний та гнучкий інструмент опису мов, транслятори для яких повинні розроблюватися, була розроблена внутрішня мова програмного середовища. Вона використовує синтаксично прості конструкції з мінімальним набором метасимволів та дозволяє за допомогою розширеної форми Бекуса–Наура (РБНФ) формулювати задачі генерації мовного процесора у звичному для даної предметної області вигляді. Приклад опису мови користувача наведено на рис. 1.

```
/* Sample grammar */

// tokens
letter = [a-zA-Z_];
digit = [0-9];
identifier = letter { letter | digit }*;

// productions
EXPR   -> TERM { '+' TERM };
TERM   -> identifier = digit;
```

Рис 1. Приклад опису мови користувача на внутрішній мові програмного середовища

Текст опису задачі генерації мовного процесора в загальному випадку складається з декількох блоків, кожен з яких визначає лексику (за допомогою регулярних виразів) або синтаксис (за допомогою РБНФ) однієї з мов користувача. На початку блоків можна використовувати конструкцію "<ключове слово> <ідентифікатор>" для іменування мов і окремих лексичних та синтаксичних розділів. Тут ключовими словами є: `language`, `token`, `production`.

Алфавіт мови програмного середовища складається із символів, з яких можуть формуватися слова мови, та символів, що повинні використовуватись тільки для запису коментарів.

До першої групи символів відносяться:

- ◊ букви латинського алфавіту: "`a-z`", "`A-Z`";
- ◊ знак підкреслення: "`_`";
- ◊ арабські цифри: "`0-9`";
- ◊ знаки операцій: "` + * ~ `";
- ◊ дужки: "` { } ( ) [ ] `";
- ◊ розділові знаки: "` ; ' | - > = `";
- ◊ символи коментарів: "` / * `".

Другу групу символів складають символи національних алфавітів та інші символи, які можуть бути відображені на екрані дисплея.

Лексику (множину термінальних слів V_T) складають службові слова та слова користувача.

До *службових слів* відносяться:

- ◊ оператори мови опису: "`language token production`";
- ◊ операції: "` - + * ~ `";
- ◊ дужки: "` { } ( ) [ ] `";
- ◊ розділові знаки: "` ; | -> = `".

До *слів користувача* відносяться:

- ◊ ідентифікатори, що позначають лексеми (для блока опису лексики) чи нетермінали та термінали (для блока опису синтаксису): "`letter EXPR`";
- ◊ атоми (ланцюжки символів в описі лексики);
- ◊ рядок, тобто послідовність літер в лапках: `'sample'`.

Таким чином, всі описані вище термінальні слова утворюють множину термінальних слів V_T = {`-`, `+`, `*`, `~`, `{`, `}`, `(`, `)`, `[`, `]`, "`;`", `|`, `=`, `->`, "`'`", "`ідентифікатор`", "`атом`", "`рядок`", "`language`", "`token`", "`production`"}.

Правила підстановки граматики мови програмного середовища генерації мовних процесорів наведено на рис. 2.

Дії, що виконує транслятор розглядуваної мови під час розбору, описані в табл. 1. Слід відмітити, що транслятор мови опису формальних граматик активно застосовує під час розбору контекстну інформацію, що дозволяє спростити мову та зменшити кількість метасимволів за рахунок додаткових знань щодо розташування наступного вхідного символу (в лексичному чи синтаксичному блоці, в лівій чи правій частині правила тощо).

Таблиця 1

Дії компілятора мови програмного середовища генерації мовних процесорів

№ правила	Дія
Перед початком розбору	Формування порожніх таблиць: лексем, правил підстановки у формі контекстно-вільної граматики, правил підстановки у формі РБНФ. Встановлення контекстної інформації
1	–
2	Встановлення контекстної інформації
3	Встановлення контекстної інформації
4	–
5	–
6	–
7	Визначення імені блоку опису мови користувача
8	Встановлення імені блоку опису мови користувача в значення по замовчуванню
9	Визначення ключового слова; встановлення контекстної інформації
10	Визначення ключового слова; встановлення контекстної інформації

Продовження табл. 1

11	Визначення ключового слова; встановлення контекстної інформації
12	Запис в таблицю лексем нового рядка: ім'я мови, ім'я лексеми, відповідний недетермінований скінченний автомат
13	Визначення імені лексеми та встановлення контекстної інформації
14	Формування фрагмента графа переходів недетермінованого скінченного автомата (правило виду A|B)
15	–
16	Формування фрагмента графа переходів недетермінованого скінченного автомата (правило виду AB)
17	–
18	Формування фрагмента графа переходів недетермінованого скінченного автомата (правило виду A^* або A^+)
19	–
20	Визначення типу операції (* або +)
21	Визначення типу операції (* або +)
22	Формування фрагмента графа переходів недетермінованого скінченного автомата (доповнення до множини атомів – ~[atoms])
23	Формування фрагмента графа переходів недетермінованого скінченного автомата (множина атомів – [atoms])
24	Формування фрагмента графа переходів недетермінованого скінченного автомата (правило виду (A))
25	Формування фрагмента графа переходів недетермінованого скінченного автомата (правило виду {A})
26	Формування фрагмента графа переходів недетермінованого скінченного автомата (ідентифікатор)
27	Формування фрагмента графа переходів недетермінованого скінченного автомата (атом)
28	Формування фрагмента графа переходів недетермінованого скінченного автомата (рядок)
29	Визначення множини атомів (вираз виду ab)
30	Визначення множини атомів (вираз виду ab)
31	Формування інтервалу символів (вираз виду [a-b])
32	Формування інтервалу символів (вираз виду [a-b])
33	Формування інтервалу символів (вираз виду [a-b])
34	Формування інтервалу символів (вираз виду [a-b])
35	Запис нового правила підстановки в поточний блок опису синтаксису
36	Запис правила підстановки виду $A \to \varepsilon$ в поточний блок опису синтаксису
37	Визначення нетерміналу в лівій частині правила підстановки та імені функції користувача, що відповідає поточному правилу підстановки; запис правила підстановки у формі РБНФ в поточний блок опису синтаксису; встановлення контекстної інформації
38	Визначення нетерміналу в лівій частині правила підстановки та імені функції користувача, що відповідає поточному правилу підстановки; запис правила підстановки у формі РБНФ в поточний блок опису синтаксису; встановлення контекстної інформації
39	Визначення нетерміналу в лівій частині правила підстановки та імені функції користувача, що відповідає поточному правилу підстановки; запис правила підстановки у формі РБНФ в поточний блок опису синтаксису; встановлення контекстної інформації
40	Запис альтернативи виду $A \to \alpha$ в множину альтернатив поточного правила підстановки; запис відповідної альтернативи у формі РБНФ

Закінчення табл. 1

41	Запис альтернативи виду $A \to \alpha$ в множину альтернатив поточного правила підстановки; запис відповідної альтернативи у формі РБНФ
42	Запис альтернативи виду $A \to \varepsilon$ в множину альтернатив поточного правила підстановки; запис відповідних альтернатив у формі РБНФ
43	Запис альтернатив виду $A \to \varepsilon$ та $A \to \alpha$ в множину альтернатив поточного правила підстановки; запис відповідних альтернатив у формі РБНФ
44	Запис двох альтернатив виду $A \to \varepsilon$ в множину альтернатив поточного правила підстановки; запис відповідних альтернатив у формі РБНФ
45	—
46	Формування поточної альтернативи у формі РБНФ
47	Аналіз ідентифікатора на приналежність до множини терміналів чи нетерміналів, та запис даного слова в кінець поточної альтернативи та відповідної множини
48	Запис термінального слова в кінець поточної альтернативи та відповідної множини
49	Формування правил контекстно-вільної граматики, що відповідають виразу (A) у РБНФ: `'[A]' → A` Поповнення множини нетермінальних слів новим нетерміналом `'[A]'`; запис відповідного нетермінального слова в кінець поточної альтернативи. Зменшення на одиницю лічильника рівня вкладеності поточної альтернативи

50	Формування правил контекстно-вільної граматики, що відповідають виразу [A] (необов'язковий елемент) у РБНФ:
	`'[A]' → A`
	`'[A]' → ε`
	Поповнення множини нетермінальних слів новим нетерміналом `'[A]'`; запис відповідного нетермінального слова в кінець поточної альтернативи.
	Зменшення на одиницю лічильника рівня вкладеності поточної альтернативи

51	Формування правил контекстно-вільної граматики, що відповідають виразу {A} (необов'язковий список) у РБНФ:
	`'[A]' → A`
	`'[A]' → ε`
	`'LIST'[A]' → 'LIST'[A]' A`
	`'LIST'[A]' → A`
	Поповнення множини нетермінальних слів новими нетерміналами `'LIST'[A]'` та `'[A]'`; запис нетермінального слова `'LIST'[A]'` в кінець поточної альтернативи.
	Зменшення на одиницю лічильника рівня вкладеності поточної альтернативи

52	Збільшення на одиницю лічильника рівня вкладеності поточної альтернативи
53	Збільшення на одиницю лічильника рівня вкладеності поточної альтернативи
54	Збільшення на одиницю лічильника рівня вкладеності поточної альтернативи
Після завершення розбору	Перевірка таблиць лексем, правил підстановки у формі контекстно-вільної граматики та правил підстановки у формі РБНФ на наявність пустих рядків. Тест на приналежність до множини терміналів чи нетерміналів для слів, ідентифікація яких під час синтаксичного розбору була неможлива. По замовчуванню слово, яке не є лексемою, визначеною в блоці опису лексики та не зустрічалося в лівих частинах правил підстановки, вважається нетермінальним, якщо перша літера його імені велика; інакше слово вважається терміналом. Альтернативний режим тестування слів на приналежність до множини терміналів чи нетерміналів вважає нетерміналами лише слова, які зустрічалися в лівих частинах альтернатив граматики

```
1)  MAIN          →  DEC_LIST
2)  DEC_LIST      →  DEC_LIST      DEC_ITEM      eol
3)  DEC_LIST      →  DEC_ITEM      eol
4)  DEC_ITEM      →  KEY_REC
5)  DEC_ITEM      →  LEX_REC
6)  DEC_ITEM      →  PROD_REC
7)  KEY_REC       →  KEYWORD       identifier
8)  KEY_REC       →  KEYWORD
9)  KEYWORD       →  kw_language
10) KEYWORD       →  kw_token
11) KEYWORD       →  kw_production
12) LEX_REC       →  LEFT_LEX      REG_EXPR
13) LEFT_LEX      →  identifier    lex_assign
14) REG_EXPR      →  REG_EXPR      or     REG_AND
15) REG_EXPR      →  REG_AND
16) REG_AND       →  REG_AND       REG_TERM
17) REG_AND       →  REG_TERM
18) REG_TERM      →  REG_TERM_S    REG_REPEAT
19) REG_TERM      →  REG_TERM_S
20) REG_REPEAT    →  lex_star
21) REG_REPEAT    →  lex_plus
22) REG_TERM_S    →  lex_tilda     [   ATOM_LIST    ]
23) REG_TERM_S    →  [   ATOM_LIST    ]
24) REG_TERM_S    →  (   REG_EXPR     )
25) REG_TERM_S    →  \{  REG_EXPR    \}
26) REG_TERM_S    →  identifier
27) REG_TERM_S    →  atom
28) REG_TERM_S    →  string
29) ATOM_LIST     →  ATOM_LIST     atom
30) ATOM_LIST     →  atom
31) ATOM_LIST     →  ATOM_LIST atom lex_range atom ATOM_LIST
32) ATOM_LIST     →  atom lex_range atom ATOM_LIST
33) ATOM_LIST     →  ATOM_LIST atom lex_range atom
34) ATOM_LIST     →  atom lex_range atom
35) PROD_REC      →  LEFT_PROD     PROD_EXPR
36) PROD_REC      →  LEFT_PROD
37) LEFT_PROD     →  identifier    identifier    arrow
38) LEFT_PROD     →  string        identifier    arrow
39) LEFT_PROD     →  identifier    arrow
40) PROD_EXPR     →  PROD_EXPR     or     PROD_AND
41) PROD_EXPR     →  PROD_AND
42) PROD_EXPR     →  PROD_EXPR     or
43) PROD_EXPR     →  or     PROD_AND
44) PROD_EXPR     →  or
45) PROD_AND      →  PROD_AND      PROD_AND_I
46) PROD_AND      →  PROD_AND_I
47) PROD_AND_I    →  string
48) PROD_AND_I    →  identifier
49) PROD_AND_I    →  LB  PROD_EXPR  )
50) PROD_AND_I    →  LAB PROD_EXPR  ]
51) PROD_AND_I    →  LFB PROD_EXPR \}
52) LB            →  (
53) LAB           →  [
54) LFB           →  \{
```

Рис. 2. Правила підстановки для граматики мови програмного середовища генерації мовних процесорів

Реалізація синтаксичного підсвічування тексту

З метою збільшення зручності використання мови опису формальних граматик для текстового редактора програмної системи генерації мовних процесорів був реалізований синтаксично-залежний процес редагування. Чуттєвість редактора до синтаксису відображається як виділення різним кольором частин тексту опису мови користувача. Колір певного ланцюжка символів залежить від його класу (табл. 2).

Таблиця 2

Кольори різних класів ланцюжків в синтаксичному підсвічуванні

Клас ланцюжка	Колір – (RGB)	Приклад
Звичайний текст	Чорний – RGB(0, 0, 0)	&&
Коментар	Зелений – RGB (0, 192, 0)	/* коментар */
Ідентифікатор	Чорний – RGB (0, 0, 0)	identifier
Ключове слово	Синій – RGB (0, 0, 255)	language
Метасимвол	Синій – RGB (0, 0, 255)	->
Рядок	Червоний – RGB (255, 0, 0)	'repeat'

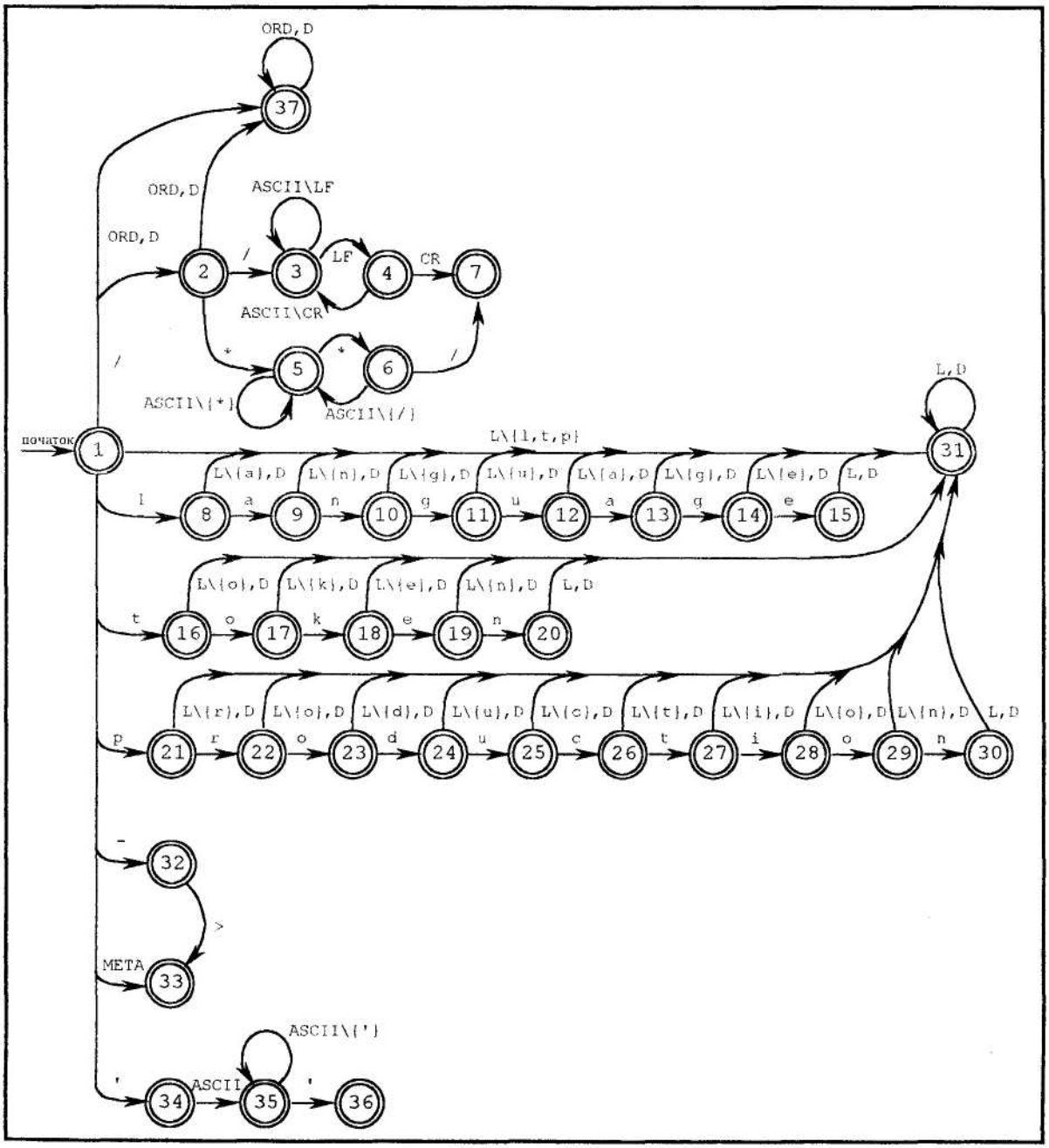

Рис. 3. Граф переходів детермінованого скінченного автомата для розпізнавання класу ланцюжка в синтаксичному підсвічуванні

Класи ланцюжків визначаються за допомогою детермінованого скінченного автомата, граф переходів якого наведено на рис. 3. Позначення множин символів алфавіту скінченного автомата розглянуто в таб. 3. Відзначимо, що для покращення продуктивності роботи автомата,

його алфавіт співпадає з множиною символів ASCII, що надає можливість використовувати відповідний код символу як один із входів таблиці переходів.

Таблиця 3

Позначення множин символів алфавіту скінченного автомата

Позначення	Множина символів
\0	'\0'
CR	повернення каретки ('\0A')
LF	переведення рядка ('\0D')
L	великі та малі латинські літери і знак підкреслення (A-Za-z)
D	арабські цифри (0-9)
META	[ ] ( ) { } * + ~ | = ;
ASCII	множина ASCII без символу '\0' ('\x01' - '\xFF')
ORD	ASCII \ (CR ∪ LF ∪ L ∪ D ∪ META ∪ {-,>,',/})

Як видно з графа, всі стани детермінованого скінченного автомата є заключними. Саме за останнім станом скінченного автомата визначається клас ланцюжка в синтаксичному підсвічуванні (табл. 4).

Таблиця 4

Визначення класу ланцюжка

Клас ланцюжка	Номери станів скінченного автомата
Звичайний текст	1, 2, 37
Коментар	3–7
Ідентифікатор	8–14, 16–19, 21–29, 31
Ключове слово	15, 20, 30
Метасимвол	32, 33
Рядок	34–36

Приклад роботи користувача в програмному середовищі та синтаксичне підсвічування для відповідного вхідного тексту наведено на рис. 4.

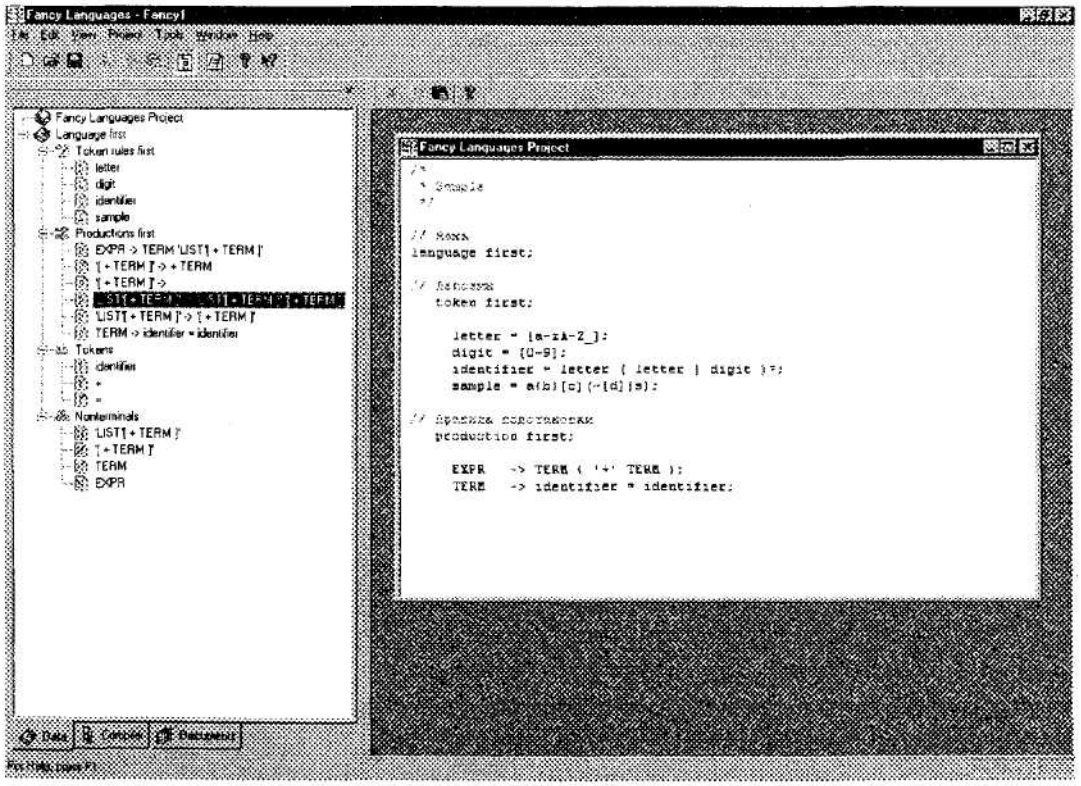

Рис. 4. Приклад роботи користувача в програмному середовищі генерації мовних процесорів "FancyCC 3.0"

ЛІТЕРАТУРА:

1. *Колодницький М.М., Левицький В.Г.* Адаптивна організація лінгвістичного забезпечення програмного комплексу "DSR Open Lab 1.0" // Праці І-ї міжн. наук.-практ. конф. з програмування "УкрПрог'98", 2–4 вересня 1998 р. – Київ, 1998. – С. 145–155.
2. *Kolodnytsky M., Ivanitsky I., Kovalchuk A., Kuryata S., Levitsky V.* "DSR Open Lab 1.0" – software system for simulation // Proceedings of 21st International Conference on Information Technology Interfaces, Pula, 1999. – P. 31.
3. *Колодницький М.М., Левицький В.Г.* Типологія архітектури інтерфейсу користувача прикладної програмної системи "DSR Open Lab 1.0". Ч. II. Лінгвістичне забезпечення // Проблемы программирования, 1999. – Вып. 3–4. (в друці)
4. *Колодницький М.М., Левицький В.Г., Рожик О.А.* Автоматизація побудови компілятора мови опису математичних моделей динамічних систем // Вісник ЖІТІ. – 1996. – № 4. – С. 138–152.
5. *Колодницький М.М., Левицький В.Г.* Сучасні інструментальні засоби автоматизованої побудови мовних процесорів // Праці II-ї міжн. наук.-практ. конф. з програмування "УкрПрог'00", 23–26 травня 2000 р. – Київ, 2000. (в друці)
6. *Колодницький М.М.* Тривимірна компонентна архітектура прикладної програмної системи "DSR Open Lab 1.0" як втілення концепцій реінженерії // Проблемы программирования, 1998. – Вып. 4. – С. 37–45.
7. *Кристиан К.* Введение в ОС UNIX. – М.: Финансы и статистика, 1985. – 320 с.
8. *Тихомиров В.П., Давыдов М.И.* ОС ДЕМОС: инструментальные средства программирования. – М.: Финансы и статистика, 1988. – 204 с.
9. *Johnson S.C.* YACC – yet another compiler-compiler. Comp. Sci. tech. rep. № 32. Bell Laboratries: Murray Hill, New Jersey, 1975.
10. *Lesk M.E.* Lex – lexical analyzer generator. Comp. Sci. tech. rep. № 39. Bell Laboratries: Murray Hill, New Jersey, 1975.
11. *Grosch J., Emmelmann H.* A Tool Box for Compiler Construction, Technical Report No. 20, Gesellschaft für Mathematik und Datenverarbeitung mbH, Forschungsstelle an der Universität Karlsruhe, 1990.
12. *Schröer F.W.* The GENTLE Compiler Construction System, R. Oldenbourg Verlag, Munich and Vienna, 1997, 142 p.
13. *Parr T.J.* Language Translation Using PCCTS and C++. A Reference Guide. Automata Publishing Company, 1993, 310 p.
14. *Tanenbaum A.S.* et al. Amsterdam Compiler Kit documentation, Rapport nr IR-90, Vrije Universiteit, Amsterdam, 1984.

ЛЕВИЦЬКИЙ В'ячеслав Георгійович – аспірант Житомирського інженерно-технологічного інституту.

Наукові інтереси:

- комп'ютерні інформаційні технології;
- моделювання і розв'язок задач за допомогою обчислювальної техніки;
- побудова компіляторів.

Подано 16.11.2000.