5. Лабораторний практикум побудови синтаксичних аналізаторів.

Граматика мови програмування визначається множиною БНФ-правил, що записані в текстовому файлі. Кожне провило обов'язково починається з першої позиції рядка. Для зручності правило можна продовжити в наступних рядках, але не з першої позиції. Нетермінал граматики - це ланцюжок літер, який починається з символу # та закінчується символом #, наприклад #програма#. Термінальні ланцюжки записуються традиційно, наприклад, begin. Альтернативи правила для зручності позначаються символом ! в першій позиції рядка, при цьому ліва частина правила опускається. Оскільки символи #, \ та ! є метасимволами при визначенні граматики, то для їх запису в термінальних ланцюжках використовується ще один метасимвол, а саме \. Правило граматики в текстовому файлі записується так:
#Pascal - програма# program (#список параметрів#) #блок# .

Лабораторний практикум другого семестру з дисципліни «Системне програмування» передбачає освоєння навиків створення власних програмних Java-проектів з використанням базового проекту, який повністю реалізований, та надається студенту у використання.

Визначимо основні положення базового проекту. Клас ***class MyLang*** забезпечує повне рішення та надає програмісту можливість замінити в екземплярі результуючого класу певне значення на власний результат та виконати проект в цілому. Якщо новостворена програмна система надає фінальний результат, який відповідає результату базового проекту, то алгоритм, який реалізований студентом, рахується вірним.

Таким чином, студенту надається «конструктор цеглин» якими можна скористатися на етапі реалізації власного алгоритму (звичайно, **не викликаючи «будівельний матеріал» базового проекту, який надає аналогічний результат**).

class MyLang

Тоді коротко, основні поля класу ***class MyLang*** наступні:

private int axioma;	// код аксіоми. В граматиці аксіома може бути не в першому правилі граматики
private boolean create;	// якщо екземпляп классу створено корректно, то true
private int LLK;	// значення LLK-контексту
private LinkedList<Node> language;	// список правил грамат
private LinkedList<TableNode> lexemaTable;	// таблиця перекодування лексем граматики
private int[] terminals;	// таблиця терміналів граматики

private int[] nonterminals;	// таблиця нетерміналів граматики
private int[] epsilonNerminals;	// таблиця ерсілон-нетерміналів граматики
private LlkContext[] termLanguarge;	// масив одно літерних словарних, побудованих на основі терміналів, тобто termLanguarge[i] – це одноелементна словарна множина, побудована на основі terminals[i]
private LlkContext[] firstK;	// массив словарних множин $first_k$, побудований выдповыдно до масиву nonterminals, тобто $first_k$[i] це множина $First_k$ для не термінала nonterminals[i].
private LlkContext[] followK;	// аналогічно попередньому поясненню
private LinkedList<LlkContext>[] LocalkContext;	// аналогічно попередньому поясненню, тобто LocalkContext[i] – це список множин $Local_k$ для нетермінала nonterminals[i].
private int[] uprTable;	// двомірний масив, який моделюється одноміпним масивом, у якого кількість стопців рівна (terminals.length+1), а рядків - nonterminals.length.

Конструктор екземплярів класу: public MyLang(String fileGramma, int llk).

Як бачимо, головний елемент (поле) класу ***MyLang*** LinkedList<Node> language – це список правил граматики. Кожне правило представляється екземпляром класу ***Node***. У загальному випадку, аксіома – не обов'язково лівий нетермінал першого правила. Вона визначається через метод public int getAxioma().

Множини $First_k$ (A_i), Followk(A_i), $First_k(w) \oplus_k Fllow_k(A_i)$ (для правила A_i -> w) – визначаються як екземпляри класу ***LlkContext***.

Множина Localk(S,A_i) визначається як ***LinkedList<LlkContext>***, тобто множина (список) елементів ***LlkContext.***

Для роботи з інформацією класу ***MyLang*** користувачу надається набір методів:

Група методів, яка забезпечує виконання алгоритмів:
public boolean isCreate() ; // Перевіряє, чи створено «complite» екземпляр класу ***MyLang***
public LinkedList<LlkContext>[] createLocalK() ;// побудова множин $Local_k(S,A_i)$
public boolean llkCondition(); // перевірка LL(k) - умови
public boolean createNonProdRools();// пошук непродуктивних правил
public boolean createNonDosNeterminals();// пошук недосяжних нетерміналів
public boolean leftRecursNonnerminal();// пошук ліво-рекурсивних нетерміналів
public boolean rightRecursNonnerminal();// пошук право-рекурсивних нетерміналів

public LlkContext[] firstK();(); // побудова множини $First_k(A_i)$,
public LlkContext[] followK();// побудова множини $Follow_k(A_i)$,
public void firstFollowK(); // побудова множини $First_k(w) +_k Follow_k(A_i)$ для всіх правил
public int[] createEpsilonNonterminals();// побудова множино е-нетерміналів
private int[] createTerminals(); // побудова множино терміналів
private int[] createNonterminals(); // побудова множино нетерміналів
public boolean strongLlkCondition() ; // перевірка сильноє LL(k)-умови
public int[] createUprTable() ; // побудова таблиці управління для LL(1)-граматики
public int[] createUprTableWithCollision(); // побудова таблиці управління для LL(1)-граматики за умови, що декілька не терміналів мають LL(1)-колізію, але граматика взагалі задовольняє сильній LL(2)-умові. Клітини таблиці управління, де є колізія позначаються (-1).

Група сервісних методів – забезпечує екстракцію даних з класу або запис даних до класу :
public int getAxioma();
public void setLocalkContext(LinkedList<LlkContext>[] localK) ; // дає можливість внести до екземпляр класу ***MyLang*** побудовану множину $Local_k$; Взагалі, в цьому проекті методи set забезпечують збереження результату в екземплярі класу ***MyLang***, а методи get – отримання результату.
public LinkedList<LlkContext>[] getLocalkContext();
public int[] getUprTable();
public LlkContext[] getFirstK();
public void setFirstK(LlkContext[] first) ;
public void setUprTable(int[] upr);
public LlkContext[] getFollowK();
public void setFollowK(LlkContext[] follow);
public int getLlkConst();
public LlkContext[] getLlkTrmContext(); // масив словарних множин для одноелементних термінвальних мов. І-та мова зберігає і-й термінал з масиву (множини) терміналів
public String getLexemaText(int code); //по коду лексеми ми визначаємо її текст
public int[] getTerminals()
public int[] getNonTerminals()
public int[] getEpsilonNonterminals()
public LinkedList<Node> getLanguarge()
public void setEpsilonNonterminals(int[] eps)

Група сервісних методів, які забезпечують друк (вивід у стандартний файл) результату:
public void printLocalk() ;
public void printTerminals();

public void printNonterminals();
public void prpintRoole(Node nod);
public void printFirstkContext()
public void printFirstFollowK()
public void printFirstFollowForRoole()
public void printFollowkContext()
public void printEpsilonNonterminals()
public void printGramma().

Class Node

Клас ***class Node*** забезпечує збереження та доступ до правил граматики у списку правил ***LinkedList<Node> language.*** Поля екземпляру класу ***Node:***

private int[] roole; // закодоване правило – це масив кодів лексем. Довжина масиву включає і лівий елемент правила. Нетермінали граматики закодовані від'ємними кодами, термінали – додатніми кодами.

private int teg; //робоче поле, доступне користувачу для маркування правил.

private LlkContext firstFollowK; // тут зберігається добуток Firstk(w)+$_k$ Followk(Ai) (для правила Ai-> w) для правила.

Для роботи з правилом граматики надаються наступні методи, їх семантика зрозуміла з попередніх визначень:

public void addFirstFollowK(LlkContext rezult);
public LlkContext getFirstFollowK() ;
public int[] getRoole();

Class LlkContext

Клас ***class LlkContext –*** відповідає за збереження словарних множин. Слово – це масив кодів лексем (int []). Е-слово має специфікацію new int[0] (мова Java допускає таку нотацію.

public LlkContext() ; // конструктор
public boolean wordInContext(int[] word) ; // слово є в словарній множині
public int[] getWord(int index) ; // отримати слово з індексом index з словарної множини. Індексація починається з нуля.
public int minLengthWord(); // мінімальна довжина слова у словарній множині
public int calcWords(); // кількість слів у словарній множині
public boolean addWord(int[] word) ; // добавляє слово до словарної множини. У випадку. Коли слово добавлено - значення методу **true**.

Class TableNode

Клас ***class TableNode*** **–** це допоміжний клас для зберігання текстів лексем граматики. Поля класу:
String lexemaText; // текст лексеми
int lexemacode; // код лексеми
static int numarator; // нумератор лексем

Маска коду не термінала 0x80000000, маска коду термінала 0x10000000. В цьому класі важливий конструктор:
public TableNode (String lexema, int lexemaType);

Блок 1 Лабораторних робіт

Реалізувати алгоритм пошуку е-нетерміналів. Результат занести до опису граматики - екземпляр класу *MyLang*;

Блок 2 Лабораторних робіт

Перевірити, чи є побудована граматика LL(1)-граматикою.

Блок 3 Лабораторних робіт

Реалізувати синтаксичний аналізатор мови Pascal методом рекурсивного спуску.

Блок 4 Лабораторних робіт

Наведений нижче перелік лабораторних робіт - це дослідницькі роботи, які передбачають вивчення додаткового матеріалу та практичних навиків попередніх розділів.

Побудувати LL(1)-граматику для update – statement мови програмування SQL.

Реалізувати синтаксичний аналізатор мови update – statements.