LL(1) algoritam

Grupe: 1, 3, 5, 7

Uputstvo:

Projektovati i implementirati LL(1) sintaksni analizator jezika definisanog zadatom gramatikom. Za leksičku analizu koristiti analizator projektovan u vežbi 1. Ukoliko je potrebno, transformisati zadatu gramatiku u LL(1) oblik.

Projektovanje sintaksnog analizatora po postupku obrađenom na računskim vežbama (u temi LL(1) sintaksni analizator) obuhvata transformaciju gramatike po potrebi, određivanje FIRST i FOLLOW skupova, proveru da je dobijena gramatika zaista LL(1) i popunjavanje sintaksne tabele. Uz rešenje zadatka je **potrebno pokazati ceo taj postupak** (ispisati rešenje na papiru).

U samom programu obavezno implementirati odgovarajuću **sintaksnu tabelu korišćenjem matrica (dovoljna je matrica celih brojeva za predstavljanje sintaksne tabele)**. Korišćenje nepreglednih **if-then-else** i **switch-case** struktura **umesto sintaksne tabele nije dozvoljeno**.

LR algoritam

Grupe: 2, 4, 6, 8

Uputstvo:

Projektovati i implementirati LR sintaksni analizator jezika definisanog zadatom gramatikom. Za leksičku analizu koristiti analizator projektovan u vežbi 1. Gramatiku nije potrebno transformisati.

Projektovanje sintaksnog analizatora po postupku obrađenom na računskim vežbama (u temi LR sintaksni analizator) obuhvata kreiranje kanoničkog skupa LR pravila, crtanje grafa prelaza automata za prepoznavanje vidljivih prefiksa, određivanje FIRST i FOLLOW skupova i popunjavanje sintaksne tabele. Uz rešenje zadatka je **potrebno pokazati ceo taj postupak** (ispisati rešenje na papiru).

U samom programu obavezno implementirati odgovarajuću **sintaksnu tabelu korišćenjem matrica (dovoljna je matrica celih brojeva za predstavljanje sintaksne tabele)**. Korišćenje nepreglednih **if-then-else** i **switch-case** struktura **umesto sintaksne tabele nije dozvoljeno**.

Grupa 1

ApplyExpression → **for ID in [** *NameList* **] apply** *Expression*
NameList → *NameList* **,** **ID** | **ID**
Expression → *Expression* **+** *Term* | *Term*
Term → **ID** | **CONST**

Grupa 2

IfStatement → **if (** *RelExpression* **) :** *Expression ElsePart*
ElsePart → **else :** *Expression*
RelExpression → *Term* **>** *Term* | *Term*
Expression → *Expression* ***** *Term* | *Term*
Term → **ID** | **CONST**

Grupa 3

ReadExpression → **read (ID in ID) do** *StatementList*
StatementList → *StatementList* **;** *Statement* | *Statement*
Statement → **ID = CONST** | **ID = ID**

Grupa 4

FunctionDeclaration → **ID (** *Parameters* **) =>** *Expression* **;**
Parameters → *Parameters* **,** *Parameter* | *Parameter*
Parameter → **ID** | **ID = CONST**
Expression → *Expression* ***** *Term* | *Term*
Term → **ID** | **CONST**

Grupa 5

CaseStatement → **case (ID) {** *WhenStatementList* **}**

WhenStatementList → *WhenStatementList WhenStatement* | *WhenStatement*

WhenStatement → **when CONST :** *Statement*

Statement → *CaseStatement* | **ID = ID ;** | **ID = CONST ;**

Grupa 6

WhileLoop → **while** *Expression* **:** *Statement* **else** *Statement*

Expression → *Expression* **or** *AndExpression* | *AndExpression*

AndExpression → *AndExpression* **and** *Term* | *Term*

Term → **ID** | **CONST**

Statement → *WhileLoop* | **ID :=** *Expression* **;**

Grupa 7

RedoLoop → **loop (** *Expression* **) {** *Statement* **redo (** *Expression* **) ;** *Statement* **}**

Expression → *Expression* **||** *AndExpression* | *AndExpression*

AndExpression → *AndExpression* **&&** *Term* | *Term*

Term → **ID** | **CONST**

Statement → *RedoLoop* | **ID =** *Expression* **;**

Grupa 8

SelectStatement → **select begin** *CaseList* **end**

CaseList → *CaseList Case* | *Case*

Case → **case ID =>** *Statement*

Statement → *SelectStatement* | **ID := ID ;** | **ID := CONST ;**