Processadors de Llenguatges

Pràctica 1 – Anàlisi lèxica

Teresa Alsinet
teresa.alsinet@udl.cat

Curs 2022-2023

Objectius

La implementació manual de la rutina d'anàlisi lèxica és una tasca costosa. L'objectiu d'aquesta pràctica és utilitzar una eina de generació automàtica d'analitzadors lèxics, les quals estan especialment dissenyades per ser utilitzades conjuntament amb una eina de generació automàtica d'analitzadors sintàctics.

Exercicis de programació

Exercici 1 (Processat del lletguatge regular constants enteres) (2 punts)
Implementeu un traductor del lletguatge regular constants enteres. El traductor llegiràuna constant entera, en format seqüència de caràcters, i la convertirà en el corresponent valor enter en notació decimal (valor enter en notació decimal).

Les constants enteres podran estar expressades en notació decimal (expressades directament sense cap símbol inicial), octal (indicat amb el caràcter o o O inicial), hexadecimal (indicat amb els caràcters ox o OX inicial), o binari (indicat amb el caràcter b o B inicial). Considereu que les constants són sense signe.
Amb el format descrit, exemples de constants enteres són els següents:

1. 568
2. 86753
3. oxAF34
4. OX3BC

5. c527
6. C777
7. b1001
8. B0001101

El valor enter en notació decimal per a cadascuna de les constants anteriors és el següent:

1. 568
2. 86753
3. 44852
4. 956
5. 343
6. 511
7. 9
8. 13

Consideracions:

- Les constants a analitzar estaran en un fitxer de test. Proporcioneu el nom del fitxer d'entrada com millor us vagi depenent del sistema escollit.
- La sortida del traductor serà únicament els valors enters en notació decimal i els podeu escriure directament a la sortida estàndard.
- Els caràcters blancs, tabuladors i nova línia actuaran com a delimitadors o separadors de constants. A més, comptabilitzeu els números de línia.
- El processat acabarà amb la marca de final d'arxiu.
- Definiu un format per a incloure línies de comentaris i ignoreu els comentaris trobats a l'entrada.
- Al detectar un error (qualsevol caràcter que no sigui un dels descrits), el traductor emetrà un missatge indicant el caràcter que l'ha produït i el número de línia del programa font on ha estat detectat. La tècnica de recuperació associada serà el mode pànic: l'analitzador lèxic despreciarà tots els símbols fins trobar un dels caràcters delimitadors (blancs, tabuladors i nova línia) o la marca de final d'arxiu.

- Les constants a analitzar estaran en un fitxer de test. Proporcioneu el nom del fitxer d'entrada com millor us vagi depenent del sistema escollit.

- La sortida del traductor serà únicament els valors enters en notació decimal i els errors detectats. La sortida es pot escriure directament a la sortida estàndard.

Exercici 2 (Lleguatge de programació) (3 punts)
Escolliu un llenguatge i, per a aquest llenguatge, implementeu una especificació lèxica que calculi el nombre d'ocurrències per a cada component o categoria del llenguatge.

En particular, comptabilitzarem el nombre de paraules reservades, identificadors, constants (enteres, reals, de caràcter, de cadena -string- ....), operadors (aritmètics, lògics, relacionals, d'assignació ....), símbols delimitadors ('(', ')', '[', ']', '', '', ',', ';' ...), símbols de separació (blancs, tabuladors, new line ...) i comentaris, que apareixen en un programa escrit en el llenguatge. Comptabilitzeu qualsevol altra component com a components auxiliars.

Consideracions:

- Per a cada categoria de components del llenguatge, indiqueu la proporció que representa respecte del total analitzats.

- Elaboreu fitxers de prova per al vostre analitzador. Proporcioneu el nom del fitxer d'entrada (prova) com millor us vagi depenent del llenguatge i entorn escollit.

- La sortida de l'analitzador (freqüència d'ocurrència per a cada categoria del llenguatge) la podeu escriure directament a la sortida estàndard.

- En aquest problema, no hi ha detecció d'errors, ja que comptabilitzem el que hem anomenat components auxiliars.

- Prepareu un fitxer README indicant el llenguatge i les característiques implementades.

Exercici 3 (Llenguatge de la Lògica Proposicional) (2 punts)
Implementeu una especificació lèxica que reconegui els components del llenguatge fòrmules proposicionals (CP0) definides sobre les variables proposicionals denotades mitjançant qualsevol lletra en majúscula, és a dir, sobre el rang [A-Z].

Consideracions:

- Els operadors a considerar són: la negació !, la conjunció $\wedge$, la disjunció $\vee$, la implicació que representarem amb el string "$->$", i la doble implicació que representrarem amb el string "$<->$".

- Les fòrmules proposicionals podran estar parentitzades, és a dir, els símbols de parèntesis "(" i ")" són elements del llenguatge.
- El caràcter nova línia actuarà com a marca de final fòrmula i la marca de final d'arxiu com a final d'entrada. El processat acaba amb la marca de final d'arxiu.
- Desprecieu els caràcters blancs i tabuladors existents a l'entrada.
- Definir un format per a incloure línies de comentaris.
- Al detectar un error, l'analitzador lèxic emetrà un missatge indicant el caràcter que l'ha produït i el número de línia del programa font on ha estat detectat. La tècnica de recuperació associada és el mode pànic: l'analitzador lèxic despreciarà tots els símbols fins el caràcter nova línia o la marca de final d'arxiu.
- Les fòrmules proposicionals a analitzar estaran en un fitxer de test. Proporcioneu el nom del fitxer d'entrada com millor us vagi depenent del sistema escollit.
- La sortida del analitzador serà únicament els possibles errors i els podeu escriure directament a la sortida estàndard.
- Prepareu un fitxer README indicant les característiques implementades.

Exemples de fòrmules lèxicament correctes són:

- $P \wedge Q <-> (!R \vee (Q-> T))$
- $P\wedge <-> (!R \vee (Q-> T))$
- $P \wedge Q <-> (!R \vee (Q-> T$
- $P \;\; Q <-> (!R \;\; (Q-> T))$

És a dir, a nivell lèxic no identifiquem possibles errors sintàctics. En canvi, **exemples de fòrmules lèxicament incorrectes són**:

- $p \wedge q <-> (!R \vee (Q-> T))$
- $P + Q * (!R \vee (Q-> T))$
- $[P \wedge Q <-> [!R \vee (Q-> T)]]$
- $P \wedge Q <=> (!R \vee (Q => T))$
- $P \wedge Q <-> (!R \vee (Q-> T));$

Exercici 4 (Lleguatge de la Lògica de Predicats) (3 punts)
La Lògica de Predicats (CP_1) es una extensió de la Lògica Proposicional que afegeix nous conceptes com quantificadors, funcions i predicats que permeten considerar l'estructura dels enunciats declaratius simples.

El llenguatge CP_1 té els següents components:

1. Un conjunt de símbols de variables denotades mitjançant qualsevol lletra en minúscula en el rang [x-z] seguida per un dígit, és a dir, $\{x1, x2, \ldots, x9, \ldots\}$.

2. Un conjunt de símbols de constants denotades mitjançant qualsevol lletra en minúscula en el rang [a-c] seguida per un dígit, és a dir, $\{a1, a2, \ldots, a9, \ldots\}$.

3. Un conjunt de símbols de predicats denotats mitjançant qualsevol lletra en majúscula en el rang [P-T] seguida per un dígit, és a dir, $\{P1, P2, \ldots, P9, \ldots\}$.

4. Un conjunt de símbols de funcions denotades mitjançant qualsevol lletra en minúscula en el rang [f-g] seguida per un dígit, és a dir, $\{f1, f2, \ldots, f9, \ldots\}$.

5. Els operadors de negació !, conjunció $\wedge$, disjunció $\vee$, implicació que representarem amb el string "$->$", i la doble implicació que representrarem amb el string "$<->$".

6. El quantificador universal que representarem amb la paraula reservada `forall` i el quantificador existencial que representarem amb la paraula reservada `exists`.

7. Les fòrmules de CP1 poden estar parentitzades i el símbol "," actua com a separador dels paràmetres en les funcions i predicats. És a dir , els símbols de puntuació $\{(,), ","\}$ són components del llenguatge.

A l'igual que en l'exercici anterior:

- El caràcter nova línia actuarà com a marca de final fòrmula i la marca de final d'arxiu com a final d'entrada. El processat acaba amb la marca de final d'arxiu.

- Despreciarem els caràcters blancs i tabuladors existents a l'entrada.

- Definirem un format per a incloure línies de comentaris i ignorarem els comentaris de l'entrada.

- Al detectar un error, l'analitzador lèxic emetrà un missatge indicant el caràcter que l'ha produït i el número de línia del programa font on ha estat detectat. La tècnica de recuperació associada és el mode pànic: l'analitzador lèxic despreciarà tots els símbols fins el caràcter nova línia o la marca de final d'arxiu.

- Les fòrmules proposicionals a analitzar estaran en un fitxer de test. Proporcioneu el nom del fitxer d'entrada com millor us vagi depenent del sistema escollit.

- La sortida del analitzador serà únicament els possibles errors i els podeu escriure directament a la sortida estàndard.
- Prepareu un fitxer README indicant les característiques implementades.

Exemples de fòrmules lèxica i sintàcticament correctes són:

- $forall\ x1\ \ forall\ x2\ (P1(x1, x2) -> \ P2(x2))$
- $forall\ x1\ (P1(x1, x2)\ <- >\ (exits\ x2\ P2(x2) \wedge P3(x1, x2) \vee\ !P2(f1(x1, x2)))$

A nivell lèxic també són correctes les fòrmules següents (són incorrectes a nivell sintàctic, però no a nivell lèxic):

- $forall\ x1\ \ forall\ x2\ exits\ (P1(x1, x2) -> \ P2(x2)P3(a4)$
- $forall\ x1\ (P1(x1, x2)\ \ forall\ (exitsx2P2(x2)!P3(x1, x2)\ forall\ !P2(f1(x1, x2)))$

En canvi, a nivell lèxic (tenint en compte les definicions del components) són incorrectes les fòrmules següents:

- $forall\ m1\ \ forall\ m2\ (P1(x1, x2) -> P2(x2))$
- $FORALL\ x1\ \ FORall\ m2\ (P1(x1, x2) -> P2(x2))$
- $forall\ x1\ exists\ x2\ (P1(x1, x2)\ *\ (exits\ x2\ P2(x2)\ +\ P3(x1, x2) \vee !P2(f1(x1, x2)))$
- $forall\ x1 :\ x1 >= 5\ \ forall\ x2\ .\ (P1(x1, x2) -> P2(x2))\ ;$

Exercici 5 Exercici opcional (Lleguatge de descripció d'AF) (0.5 punts)
Aquest problema té per objectiu definir un llenguatge de descripció d'un autòmat finit i implementar l'analitzador lèxic del llenguatge. La sortida serà la seqüència de tokens identificats a l'entrada. Prepareu un fitxer README indicant les característiques implementades.

Lliurament

La documentació a lliurar per a cada exercici de programació és la següent:

1. Especificació lèxica. Per implementar la solució podeu utilitzar l'eina de generació d'analitzadors lèxics que més us agradi: lex o flex (eina per generar analitzadors lèxics en C i C++), Jlex (eina per generar analitzadors lèxics en Java), PLY (implementació de lex en Python), Alex (eina per generar analitzadors lèxics en Haskell), camllex (eina per generar analitzadors lèxics en OCaml).

2. Mòduls auxiliars emprats en la implementació de la solució.
3. Joc de proves utilitzat per validar l'exercici.
4. Un fitxer README amb les particularitats pròpies de la vostra implementació.
5. Important: En cas que la vostra implementació estigui basada en codi de tercers, per exemple disponible a GitHub, ho heu d'indicar al README.

Lliurament de la pràctica al `cv.udl.cat` dins d'activitats. Lliurar un únic arxiu que agrupi tots els exercicis que presenteu, mitjançant la utilitat `tar`. Per a cada exercici: tots els fitxers font, el joc de proves utilitzat per a la validació i el fitxer README.

Avaluació

- La pràctica la podeu realitzar de manera individual o en grups de 2 o 3 persones. Indicar els components del grup.
- El pes d'aquesta pràctica és d'un 20% sobre la nota final de l'assignatura.
- La data límit per lliurar la pràctica és el **14 de març per a l'avaluació contínua**. Pels que no la tingueu acabada el 14 de març, la podeu lliurar fins el **28 de març** (data de l'examen del 1r parcial de l'assignatura).
- Cada estudiant avaluarà la seva participació i grau de satisfacció en el desenvolupament de la pràctica a:

  Autoavaluació de la Pràctica 1