Pràctica de Compilador 2020-2021
Compilador de LANS

September 28, 2020

Instruccions i criteris d'avaluació

Les pràctiques conten un 40% de la nota total de l'assignatura, i la nota mínima per aprovar l'assignatura és 4.5/10. El 100% de la nota de pràctiques de l'assignatura correspondrà al resultat d'aquesta pràctica.

La pràctica s'ha de desenvolupar en parelles. Hi haurà dues dates d'entrega via Moodle:

- 30/11/2020: Entrega de la part lèxica i sintàctica. Si no es fa aquesta entrega o no es compleixen raonablement les especificacions es descontarà fins a 1 punt de la nota final de pràctiques. Després de la correcció de l'entrega s'han de corregir els possibles errors que s'hagin detectat.
- 21/01/2021: Entrega final. Després de l'entrega via Moodle s'haurà de defensar la pràctica presencialment, dia i hora a determinar per cada grup.

La nota màxima és 10, i es repartirà de la següent manera:

- Anàlisi lèxic: fins a 1 punt. S'ha de fer tot per aprovar.
- Anàlisi sintàctic: fins a 2 punts. S'ha de fer tot per aprovar.
- Anàlisi semàntic: fins a 2 punts part obligatòria, fins a 1.5 punt part opcional.

- Generació de codi: fins a 2 punts part obligatòria, fins a 1.5 punt part opcional.
- És obligatori per aprovar completar l'anàlisi semàntic i generació de codi de:
  - Expressions (tots els operadors) i assignació
  - Operacions de lectura i escriptura
  - Condicional
  - Bucles
- Per optar a la màxima puntuació:
  - Àlies
  - Vectors
  - Tuples
  - Accions i funcions

Compilador LANS

El *Llenguatge d'Alt Nivell Senzill* (LANS) és un llenguatge de programació inspirat en el pseudocodi imperatiu que s'ha utilitzat durant molts anys en aquesta universitat. L'objectiu d'aquesta pràctica és el desenvolupament d'un compilador que rebrà com a entrada un codi escrit en *Llenguatge d'Alt Nivell Senzill* (LANS) i ens generarà un fitxer Bytecode (.class), que podrà ser executat amb la *Java Virtual Machine* (JVM). El propi compilador també es desenvoluparà en Java. La pràctica es desenvoluparà utilitzant ANTLR4 (*ANother Tool for Language Recognition*), que ens sistematitzarà la construcció dels analitzadors lèxics i sintàctics.

1 Tipus de dades

En LANS tenim *tipus bàsics* de dades i *tipus definits*.

1.1 Tipus bàsics

Els tipus bàsics de dades que tenim en LANS són els següents.

enter Representa un nombre enter.

real Representa un nombre real. El separador de decimal és el punt, i podem utilitzar notació científica. Exemples vàlids de nombres reals: 0.1, 2.10, 3.1416, 6.023E23, 12E-5.

car Representa un caràcter. L'expressarem entre cometes simples, exemple: 'a'. Com a caràcter només tenim lletres minúscules i majúscules, nombres, i els símbols de puntuació i altres caràcters ASCII bàsics.

boolea Representa un Booleà. Pot prendre com a valor `cert` o `fals`.

data Representa una data. Aquesta data serà posterior o igual al 1 de gener de 1980. El format serà: dd/mm/aaaa. Per exemple: 01/01/1980, 31/12/2020. El dia sempre tindrà dos dígits, el mes també dos i l'any 4.

A més a més, també podrem representar valors constants *string* entre cometes dobles, exemple: "Aixo es un string". Igual com en el cas dels caràcters, els strings només poden contenir els caràcters ASCII bàsics. Només utilitzarem els strings com a paràmetre de les instruccions d'escriptura tal i com s'explica a la Secció 6.6. No existeixen les variables de tipus string ni podem fer cap operació amb un string.

1.2 Tipus definits

Amb LANS podem definir nous tipus de dades, que poden ser tuples, vectors o àlies per a tipus bàsics. Els tipus definits es poden utilitzar per definir variables del programa principal així com de qualsevol acció o funció. Els paràmetres de les accions i funcions també poden ser de tipus definits, però el valor de retorn de les funcions no.

2 Estructura general d'un programa LANS

Un programa LANS s'escriu en un sol fitxer —no tenim importacions de codi d'altres fitxers—, i sempre especifica un nom de programa. El fitxer

que generarà el compilador rebrà el mateix nom que el programa compilat. Per exemple, el programa *HelloWorld* es compilarà per defecte en un fitxer *HelloWorld.class*.

L'estructura general d'un fitxer LANS conté els següents element (l'ordre en que apareixen ha de ser aquest):

⟨bloc de declaracio de constants⟩?
⟨bloc de declaracio de tipus⟩?
⟨bloc declaracio d'accions i funcions⟩?
`programa` *nom_programa*
⟨bloc de declaracio de variables⟩?
⟨sentencia⟩∗
`fprograma`
⟨implementacio de les accions i funcions⟩?

2.1 Bloc de declaració de constants

El bloc de declaració de constants té la forma següent:

`constants`
⟨⟨tipus basic⟩ *id_constant* := ⟨valor tipus basic⟩;⟩∗
`fconstants`

Les constants són accessibles des del programa principal així com des de qualsevol acció i funció, i evidentment no es poden modificar.

2.2 Bloc de declaració de tipus

El bloc de declaració de tipus té la forma següent:

`tipus`
⟨declaració de tipus nou⟩∗
`ftipus`

Un tipus nou pot ser un àlies per un tipus bàsic:

nom_tipus_nou : ⟨tipus bàsic⟩;

Un tipus nou també pot ser un vector de tipus bàsic d'una dimensió:

nom_tipus_nou : `vector` ⟨tipus bàsic⟩ `mida` ⟨enter_mida⟩ ⟨ `inici` ⟨enter_inici⟩⟩?;

El valor obligatori *enter_mida* és un nombre enter que determina la mida del vector. El valor opcional *enter_inici* és un nombre enter que indica quin és el primer índex vàlid del vector (per defecte 0).

Finalment, un nou tipus també pot ser una tupla de camps de tipus bàsic:

nom_tipus_nou : `tupla`
⟨*id_camp*⟩ : ⟨tipus bàsic⟩+
`ftupla`;

Fixeu-vos que les tuples han de tenir un camp com a mínim.

2.3 Bloc de declaració d'accions i funcions

En el bloc de declaració d'accions i funcions, podem declarar tantes accions i/o funcions com vulguem (o cap).

Les declaracions d'una acció i una funció tenen la forma següent:

`accio` *nom_accio*(⟨parametres formals⟩?);

`funcio` *nom_funcio*(⟨parametres formals⟩?) `retorna` ⟨tipus bàsic⟩;

Els parametres formals tenen la forma següent:

⟨⟨`ent`|`entsor`⟩? *id_parametre*:tipus ⟨,⟨`ent`|`entsor`⟩? *id_parametre*:tipus ⟩*⟩+

El modificador de paràmetre ens indica si la variable és d'entrada (`ent`) o si és d'entrada sortida (`entsor`). Si no s'especifica el modificador, el paràmetre és d'entrada. Les funcions només poden rebre paràmetres d'entrada.

2.4 Bloc d'implementació d'accions i funcions

En el bloc d'implementació d'accions i funcions s'han d'implementar les accions i funcions declarades. És necessari que totes les accions i funcions

declarades estiguin implementades, i viceversa. Es poden sobrecarregar accions i funcions. L'ordre en que apareixen implementades no ha de ser necessàriament l'ordre en que apareixen definides.

La implementació de les accions i funcions tenen la forma següent:

`accio` *nom_accio*(⟨paràmetres formals⟩?)
⟨bloc de declaracio de variables⟩?
⟨sentència⟩*
`faccio`

`funcio` *nom_funcio*(⟨paràmetres formals⟩?) `retorna` ⟨tipus bàsic⟩
⟨bloc de declaracio de variables⟩?
⟨sentència⟩*
`retorna` ⟨expr tipus bàsic⟩;
`ffuncio`

2.5 Bloc de declaració de variables

El bloc de declaració de variables té la forma següent:

`variables`
⟨⟨*id_variable*⟩⟨, *id_variable*⟩* : ⟨tipus bàsic⟩⟩+
`fvariables`

Les variables només són accessibles des del programa principal o acció o funció on s'hagin declarat.

3 Identificadors

Els identificadors (noms de constants, variables, tipus, accions, funcions i camps de tupla) han de ser una paraula sense espais que només pot contenir caràcters de la `a` a la `z`, ja sigui majúscules o minúscules, dígits `0-9`, o guió baix _. No poden començar amb un dígit.

4 Comentaris

Els comentaris en LANS són com els de C. Tenim comentaris d'una sola línia que comencen amb //, i s'acaben quan trobem un salt de línia. També podem fer comentaris multi-línia, que comencen amb `/*` i acaben quan trobem **la primera ocurrència** de `*/`.

5 Expressions

Una expressió pot ser:

- Un valor constant de tipus bàsic
- Una constant
- Una variable
- Un accés a tupla
- Un accés a vector
- Una crida a una funció
- Una operació sobre una o vàries expressions

Els operadors que acceptem són:

- + , - : suma, resta. Definits sobre enters i reals. També podrem fer sumes i restes de data i enter. Exemple : 31/12/2020+7 donarà 07/01/2021. (Si el resultat es més petit que 01/01/1980 o més gran que 31/12/9999, donarà 01/01/1980).
- *: suma, resta, multiplicació. Definits sobre enters i reals.
- /: divisió real. Definida entre enters i reals. El resultat sempre és un real.
- \, %: divisió entera, resta de divisió entera (mòdul). Definits entre enters, el resultat és un enter.
- $\sim$: menys unari. Definit sobre enters i reals. Aquest operador canvia de signe l'expressió de la seva dreta.

- == , !=: igualtat, desigualtat. Definit sobre tots els tipus bàsics. El resultat és Booleà.
- <, <=, >, >=: més petit, més petit igual, més gran, més gran igual. Definits sobre tots els tipus bàsics. El resultat és Booleà.
- `no` , & , |: negació lògica, and lògica, or lògica. Definits sobre Booleà i el resultat és Booleà.

A més tenim la expressió *if-then-else*, igual com en llenguatge C, que s'expressa com a:

`condicio` ? `exprcert` : `exprfals`

On `condició` ha de ser una expressió de tipus `boolea`, `exprcert` i `exprfals` han de tenir el mateix tipus, i el valor de retorn serà `exprcert` quan `condicio` valgui `cert`, i `exprfals` quan `condicio` valgui `fals`.

Les promocions d'enter a real són admeses i automàtiques quan sigui convenient. Per exemple, són expressions correctes "3.5 + 1", "2==4.7", "a ? 3 : 7.8", i en aquest exemples promocionem 1, 2 i 3 d'enter a real. No són admeses altres conversions entre tipus bàsics.

Els vectors s'indexen amb []:

id_variable[⟨expr_entera⟩]

Els camps de les tuples s'accedeixen amb un punt:

id_variable.nom_camp

Les funcions s'accedeixen passant els paràmetres reals (expressions) entre parèntesis i separats per coma:

nom_funcio(⟨⟨expr⟩⟨,⟨expr⟩⟩*⟩?)

La prioritat dels operadors és, de més a menys prioritari:

no ∼
* / \ %
+ −
== =! < <= > >=
& |
if−then−else

En cas d'operadors d'igual prioritat, tindrem associativitat per l'esquerra, exemple: $4 - a + b - 3 = ((4 - a) + b) - 3$. Podem modificar prioritats i associativitat posant expressions entre parèntesis.

6 Sentències

Una sentència pot ser una assignació, un condicional, un bucle de tipus *per*, un bucle de tipus *mentre*, una crida a una acció, o una instrucció de lectura/escriptura.

6.1 Assignació

Una assignació té la forma següent:

nom_variable `:=` ⟨expr⟩;

Podem assignar posicions d'un vector i camps d'una tupla, però no assignarem tot un vector ni tota una tupla. Podem assignar un enter a un real, però no al revés.

6.2 Condicional

Un condicional té la forma següent:

`si` ⟨expr_booleana⟩ `llavors`
⟨sentencia si condició certa⟩*
⟨`altrament`
⟨sentencia si condició falsa⟩*
⟩?
`fsi`

6.3 Per

El bucle *per* itera una variable entera en un rang inclusiu pels dos extrems:

`per` ⟨variable_entera⟩ `de` ⟨expressio_entera⟩`fins` ⟨expressio_entera⟩ `fer`
⟨sentencia⟩*
`fper`

La variable entera que s'itera ha d'haver estat declarada prèviament.

6.4 Mentre

El bucle *mentre* repeteix les instruccions mentre la condició sigui certa:

`mentre` ⟨expressio_booleana⟩ `fer`
⟨sentencia⟩*
`fmentre`

6.5 Crida a acció

Una acció es crida de la mateixa manera que una funció:

nom_accio(⟨⟨expr⟩⟨,⟨expr⟩⟩*⟩?);

6.6 Instruccions de lectura i escriptura

Tenim la instrucció de lectura `llegir` i les instruccions d'escriptura `escriure` i `escriureln` com a accions integrades al llenguatge.

La instrucció `llegir` rep un sol paràmetre de sortida que és una variable de tipus bàsic on guardarem el valor llegit per teclat:

`llegir`(⟨id_variable⟩);

La instrucció `escriure` rep un nombre variable de paràmetres (almenys n'ha de rebre un), i escriu per pantalla la concatenació dels diferents valors rebuts:

`escriure`(⟨expr⟩⟨,⟨expr⟩⟩*);

La instrucció `escriureln` és semblant a la instrucció `escriure`, però al final de tot escriu un salt de línia. A més a més, la instrucció `escriureln` pot no rebre cap paràmetre. Tan `escriure` com `escriureln` poden rebre expressions de tipus string.