Compiladors (CL)
GEI (2021-22)

Pràctica 1: Calculadora en MiniJulia

Objectiu

Fer la part frontal d'un compilador que interpreti programes senzills.

Descripció

Implementar una calculadora que permet manipular dades de tipus enters, reals, cadenes i booleans. També ha d'admetre vectors i matrius d'enters i reals. El llenguatge seleccionat és un subconjunt de Julia, que anomenem MiniJulia.

Aprenentatges

Comprendre els conceptes bàsics d'un compilador: taula de símbols, anàlisi lèxica, anàlisi sintàctica, atributs, anàlisi semàntica i comprovació de tipus. Utilització conjunta de flex, bison i una taula de símbols (symtab, pròpia, o d'una altra llibreria).

Llenguatge font

Literals i comentaris

- S'admeten literals enters (e.g. `123`, `045`), reals (e.g. `3.1416`, `0.5e-04`, `1.2345e3`), cadenes (e.g. `"Hola"`) i booleans (`true`, `false`).
- Les cadenes estan delimitades per dobles cometes (`"`), i no poden ocupar més d'una línia.
- Poden haver comentaris com en Julia, és dir, comentaris que comencen amb `#` fins a final de línia, i comentaris que es poden estendre vàries línies, iniciats amb `#=` i finalitzats amb `=#`.

Identificadors, variables i tipus

- Els identificadors són "case sensitive", i estan formats per una lletra seguida per un nombre arbitrari de lletres, dígits i el caràcter "_" (underscore); els "_" no poden estar ni al principi ni al final de l'identificador, ni pot haver dos seguits. Es recomana utilitzar "snake case" per a tots els identificadors, però no és obligatori.
- Les variables i expressions poden ser d'un tipus simple:
  - `Int64`: enter
  - `Float64`: real
  - `String`: cadena
  - `Bool`: booleà

- Les variables i expressions també poden ser d'un tipus compost vector o matriu:
  - `Vector{Int64}`: vector d'enters
  - `Vector{Float64}`: vector de reals
  - `Array{Int64,2}`: matriu d'enters
  - `Array{Float64,2}`: matriu de reals
- Exemples:

```
i = 10                  # enter
r = 0.5                 # real
s = "Hola"              # cadena
b = true                # booleà
v_i = [10; 20; 30]      # vector d'enters longitud 3
v_r = [1.0; 2; 3.0]     # vector de reals longitud 3
m_i = [1 2 3; 4 5 6]    # matriu d'enters 2x3
m_r = [1 2.2 3]         # matriu de reals 1x3
A = [1.1 1.2; 2.1 2.2; 3.1 3.2]   # matriu 3x2
```

- Com es veu als exemples, un vector o matriu és de reals si almenys un element és real. El punt i coma separa fileres, i l'espai en blanc separa columnes.
- L'accés a components de vectors i matrius es fa posant l'índex (o índexs separats per comes) entre claudàtors. Per exemple:

```
v_i[2]                  # valor 20
m_r[1, 3]               # valor 3.0
A[3, 1]                 # valor 3.1
```

Expressions aritmètiques

- Les expressions aritmètiques poden estar formades per literals, variables, components (de vectors i matrius) i operadors. També poden contenir crides a funcions. La llista d'operadors existents és:
  - Suma (`+`): s'aplica a enters, reals, vectors i matrius
  - Resta (`-`): s'aplica a enters, reals, vectors i matrius
  - Producte (`*`): s'aplica a enters, reals, vectors i matrius
  - Divisió (`/`): s'aplica a enters i reals
  - Mòdul (`%`): s'aplica a enters
  - Potència (`^`): s'aplica a enters i reals
  - Concatenació (`*`): s'aplica a cadenes
- Les operacions entre matrius i/o vectors donaran error si les dimensions no són les correctes, segons les definicions estàndard de suma i producte.
- S'admet com a operació el producte d'un enter o real per una matriu.
- Quan s'operen expressions aritmètiques del mateix tipus, el resultat també ho és, amb les següents excepcions:
  - La divisió d'enters té com a resultat un real
  - Enter operat amb real retorna un real; això és vàlid en tots els cassos, incloent les operacions amb vectors i matrius
- L'ordre de precedència dels operadors aritmètics binaris és: potència més precedència que producte i divisió, producte igual precedència que divisió, aquests major precedència que suma i resta, i suma igual precedència que resta.
- Els operadors aritmètics unaris (manteniment i canvi de signe) tenen igual precedència que la suma i la resta binàries.
- Es poden utilitzar parèntesis per a fer operacions sense seguir les prioritats predeterminades.

- Exemples:
  - `5 + i * 5^3 # 1255`
  - `i / 2 # 5.0`
  - `s * ", " * s # "Hola, Hola"`
  - `i + A[3, 2] / v_i[1] # 10.32`
  - `m_i[i - 8, m_i[1, 3]] # 6`
  - `-7 - (i + 7.0) # -24.0`
- Existeixen tipus diferents que són estructuralment equivalents. Quan això passa, els valors d'aquests tipus es poden utilitzar de forma totalment intercanviable en les operacions:
  - Una matriu `1x1`, un vector de longitud `1` i un número són estructuralment equivalents
  - Un vector de longitud `n` és estructuralment equivalent a una matriu columna de dimensions `nx1`
- Exemples d'operacions amb vectors, matrius i funcions predeterminades (descrites posteriorment):
  - `m_i * (v_i - 5 * v_r) # [70.0; 160.0]`
  - `3 * ones(Int64, 2, 2) # [3 3; 3 3]`
  - `transpose(v_i) + m_r # [11.0 22.2 33.0]`
  - `transpose(v_r) * v_r # [14.0]`

Expressions booleanes

- Les expressions booleanes es formen mitjançant els operadors relacionals (`>`, `>=`, `<`, `<=`, `==`, `!=`) aplicats a expressions aritmètiques, i mitjançant els operadors booleans (`!`, `&&`, `||`) aplicats a expressions booleanes. El significat de tots aquests operadors és el mateix que en d'altres llenguatges, com C, C++, Java, Julia, etc.
- L'ordre de precedència dels operadors booleans és: `!` major precedència que `&&`, i `&&` major que `||`.
- L'avaluació de les expressions booleanes es pot fer tant en curtcircuit com sense curtcircuit. Per simplicitat, es recomana fer-ho sense curtcircuit. En curtcircuit, es deixa d'avaluar la resta d'una expressió booleana tan bon punt es coneix el resultat de tota l'expressió. Per exemple, si sabem que `a` és més gran que `b`, aleshores l'expressió booleana `a > b || c < d` segur que és certa independentment del valor de `c < d`, i per tant no cal avaluar `c < d`. Sense curtcircuit s'avaluaria tota l'expressió encara que no fes falta.
- La implementació de les precedències entre operadors (tant aritmètics com booleans) s'ha de fer a través de la definició de la gramàtica; està prohibit utilitzar els operadors `%left` i `%right` que proporciona el bison.
- Exemples:
  - `i == 10 && (i + 10) >= 20 # true`
  - `v_i[1] < v_i[2] && false # false`
  - `s * s == "HolaHola" # true`
  - `b || !b # true`
  - `(b || !b) && (b && !b) # false`
  - `b || !b && b && !b # true`

Sentències

- Les sentències poden ser: expressions aritmètiques, expressions booleanes o assignacions.
- Totes les sentències ocupen una línia.
- Les assignacions són del tipus

  id = expressió

  on l'expressió pot ser aritmètica o booleana, i *id* és un identificador.
- En una assignació, el tipus de l'identificador *id* és igual al tipus del resultat de l'expressió corresponent.
- Els identificadors que apareguin en una expressió qualsevol han d'haver estat prèviament inicialitzats, ja que cal conèixer els seus tipus.

Funcions predeterminades

- Existeix una sèrie de funcions predeterminades que es poden utilitzar en expressions aritmètiques :
  - `div(i, j)`: retorna la divisió entera entre els enters `i` i `j`
  - `length(a)`: retorna el nombre d'elements d'un vector o matriu `a` (ja sigui d'enters o de reals), o la longitud d'una cadena `a`
  - `size(a)`: retorna un vector `v` (ja sigui d'enters o de reals) amb totes les dimensions del vector o matriu `a`
  - `zeros(T, n, m)`: retorna una matriu d'elements de tipus `T` (`Int64` o `Float64`) i dimensions `nxm`, inicialitzada amb zeros
  - `ones(T, n, m)`: retorna una matriu d'elements de tipus `T` (`Int64` o `Float64`) i dimensions `nxm`, inicialitzada amb uns
  - `transpose(a)`: retorna la transposada de la matriu o vector `a`

Programa

- La sintaxi d'un programa és:

  llista_de_sentències

  on cada sentència s'inicia en una línia nova, i pot ser una assignació, una expressió aritmètica o una expressió booleana.

Opcions

- Podeu afegir opcions diverses:
  - Registres: els anomenats *struct* en C i C++, *Struct* en Julia, *record* en Ada i Pascal
  - Tensors: generalització de vectors i matrius, que poden tenir tres o més índexs, i notacions i funcions per a la seva manipulació
  - Llistes: taules unidimensionals però amb elements que poden ser de tipus heterogenis, i funcions per a la seva manipulació (append, pop, merge, etc.)
  - Tuples: semblants a les llistes, però immutables, com les tuples de python o Julia
  - Noves funcions i operadors: funcions trigonomètriques, producte d'enter per cadena, extreure subcadena, funcions de format de cadenes, inversió de matrius, etc.
  - Constants predeterminades: *pi*, *e*, etc.

- Nous tipus de números: binaris i/o hexadecimals (amb les seves operacions), complexes (com a Julia o Matlab), quaternions, etc.
- Qualsevol altre opció que es pugui encabir dins del concepte de calculadora

Entrades i sortides

- L'entrada del compilador ha de ser un arxiu, d'extensió .jl (*nom.jl*), amb un programa escrit en MiniJulia, segons la descripció del llenguatge proporcionada en aquest document.
- La sortida s'ha de posar en un arxiu de text amb extensió .txt (*nom.txt*), i també s'ha d'enviar a la consola (*stdout*).
- Els noms dels arxius d'entrada i sortida s'han de poder passar per línia de comandes, en algun dels següents formats:
  - `$ ./nom_compilador nom.jl nom.txt`
  - `$ ./nom_compilador nom`
  - `$ ./nom_compilador -i nom.jl -o nom.txt`
- Les sortides de cada tipus de sentència és la següent:
  - Assignacions: mostraran el nom, el seu tipus i el seu valor.
  - Expressions aritmètiques: mostraran el seu tipus i el seu valor.
  - Expressions booleanes: mostraran el seu tipus i el seu valor.
- Convé anar guardant en un arxiu de log cada producció de la gramàtica reconeguda, per així controlar el progrés i correcció de les anàlisis lèxica i sintàctica. També es pot guardar al mateix arxiu qualsevol altre missatge informatiu que cregueu convenient.
- S'ha d'evitar barrejar la sortida de la calculadora (expressions i assignacions) de la sortida de log.
- Detecció d'errors: Quan hi ha un error al codi font s'ha d'emetre un missatge d'error indicant la posició actual a l'arxiu font, i si l'error és lèxic, sintàctic o semàntic. Mentre més informació es doni dels errors, millor.

Lliurament

- Aquesta pràctica és fa en parelles, encara que s'admet fer-la de forma individual.
- El lliurament es farà via Moodle, en les dates indicades.
- Cal lliurar un únic arxiu comprimit (zip, rar, gz, bz, etc.) que contingui:
  1. Tot el codi font (sense arxius generats en el procés de compilació)
  2. Exemples i la seva sortida corresponent
  3. Scripts o Makefile (`make`, `make clean`, `make examples`)
  4. Documentació: preferiblement en format pdf. Ha de contenir:
     - Instruccions per la compilació i execució
     - Llistat de les funcionalitats obligatòries implementades
     - Llistat de les funcionalitats opcionals implementades
     - Llistat de les limitacions de funcionament
     - Descripció extensa de tot allò que vulgueu destacar de la vostra pràctica: detalls de les funcionalitats opcionals, decisions de disseny, etc.
     - Contribució de cadascun dels membres a la pràctica
- El nom de l'arxiu lliurat ha de ser del tipus:
  - `PR1-Nom1Cognom1-Nom2Cognom2.zip`.