Compiladors (CL)
GEI (2021-22)

Pràctica 2: Compilador de MiniJulia

Objectiu

Fer les parts frontal i dorsal d'un compilador per al llenguatge MiniJulia.

Aprenentatges

Generació de codi intermedi per a expressions aritmètiques i subprogrames.

Introducció

El llenguatge de partida és el mateix MiniJulia descrit a la Pràctica 1. Aquest llenguatge s'amplia en aquesta Pràctica 2 amb la incorporació de procediments i funcions, i amb estructures de flux de control a la Pràctica 3. Per simplificació, també s'eliminen (o es fan optatives) certes funcionalitats de la calculadora (Pràctica 1).

Donat que ara l'objectiu no és fer una calculadora sinó un compilador, amb sortida que serà codi de tres adreces (C3A), hi ha diferències importants en el tractament de les instruccions. La descripció detallada del C3A es fa en un document separat; en la resta d'aquest document es suposa que es coneixen perfectament tots els detalls del C3A.

Diferència conceptual respecte la Pràctica 1

- En la calculadora de la Pràctica 1, les variables sempre tenien un valor conegut (o acabat de calcular) en tots els llocs del codi on apareixien.
- Això no es cert en el codi real. Per exemple, si una variable apareix dins d'un bucle, pot tenir valors diferents a cada volta del bucle. Per tant, el compilador no pot saber el seu valor, i el màxim que pot fer és generar les instruccions (codi intermedi o codi objecte) per a calcular i emmagatzemar el valor actual.
- Com a exemple, si tenim això en codi font

  ```
  j = 3 + i
  k = j + 3 * 7
  ```

  s'ha de suposar que en la primera instrucció no es coneix el valor actual de la variable `i`, i en la segona no es coneix el valor de `j`. La sortida corresponent en C3A podria ser

  ```
  115:  $t03 := 3 ADDI i
  116:  j := $t03
  117:  $t04 := 3 MULI 7
  118:  $t05 := j ADDI $t04
  119:  k := $t05
  ```

  o millor

  ```
  115:  $t03 := 3 ADDI i
  116:  j := $t03
  117:  $t04 := j ADDI 21
  118:  k := $t05
  ```

S'ha suposat que es coneix que la variable `i` és de tipus enter, i que ja s'han generat les 114 proposicions de C3A anteriors. La diferència entre els dos fragments de C3A és que en el segon el compilador no ha generat codi per a calcular el producte `3 * 7`, ja que aquesta operació sí que es pot fer en temps de compilació.

Llenguatge font

A continuació es detallen les diferències en el llenguatge font respecte la Pràctica 1.

Literals i comentaris

- El tractament de cadenes passa a ser optatiu; en cas de mantenir les cadenes, la seva representació lèxica és la mateixa. També desapareixen els booleans.

Identificadors, variables i tipus

- Es manté l'existència de tipus simples numèrics (enters i reals) i compostos (vectors i matrius), però s'eliminen els booleans, i el tractament de cadenes passa a ser optatiu. També s'admet de forma opcional la possibilitat de tenir tipus enters i reals de diferents mides, és dir, enters i reals de 32 bits i de 64 bits.

Expressions aritmètiques

- Les operacions amb cadenes (concatenació), i amb vectors i matrius (suma, resta i producte) passen a ser optatives. Segueix essent obligatori la possibilitat d'accedir a components de vectors i matrius.
- Poden incloure crides a funcions predeterminades o pròpies.

Expressions booleanes

- No hi ha expressions booleanes. L'avaluació d'expressions booleanes s'ha de fer en curtcircuit, utilitzant la tècnica del back-patching. Com el back-patching només es demana a la Pràctica 3 per a implementar totes les sentències de flux de control, és necessari eliminar-les d'aquesta Pràctica 2.

Sentències

- A les sentències anteriors (expressions aritmètiques i assignacions) s'afegeixen les crides a procediments i les sentències de retorn.
- Les crides a procediments i funcions (també anomenats conjuntament com a subprogrames) tenen la sintaxi

  nom_subprograma (*llista_de_paràmetres_actuals*)

  on la llista de paràmetres actuals està formada per expressions aritmètiques separades per comes. Aquesta llista podria estar buida si el subprograma no té paràmetres. Es posaran exemples de declaració i crida de procediments i funcions en la següent secció.
- Les sentències de retorn només poden aparèixer dins de procediments i funcions, i també es descriuen en la següent secció. N'hi ha dues de diferents, una pels procediments i l'altra per a les funcions.

Procediments i funcions

- Les crides a funcions ja s'havien considerat a la Pràctica 1, però només per a funcions predeterminades. Ara s'afegeix la possibilitat de crear funcions pròpies, així com crear i cridar procediments.
- A diferència de les sentències, els procediments i funcions ocupen múltiples línies.
- La sintaxi dels procediments és de la forma

  function *nom_procediment* (*llista_de_paràmetres_formals*)
    llista_de_sentencies
  end

  on la llista de paràmetres formals està formada per parells

  nom_param::*tipus_param*

  separades per comes. La llista de paràmetres formals pot estar buida.
- La sintaxi de les funcions és de la forma

  function *nom_funció* (*llista_de_paràmetres_formals*)::*tipus_retorn*
    llista_de_sentencies
  end
- Dins del cos dels procediments hi pot haver sentències de retorn directe:

  return
- Dins del cos de les funcions hi pot haver sentències de retorn de valor:

  return *expressió*
- No es permet tenir subprogrames encaixats, és dir, no es poden definir subprogrames dins d'altres subprogrames (per això s'ha preferit no considerar-los com a sentències, tal com es fa en altres llenguatges de programació).
- Exemple:

```
function swap(v::Vector{Int32})
  temp = v[1]
  v[1] = v[2]
  v[2] = temp
end

function mean(a::Float64, b::Float64)::Float64
  return (a + b) / 2
end

v = [1; 2; 3; 4]
swap(v)
x = 17.0
i = 1
z = x + mean(x - 7, v[3 + i] * 3.5)
```

Funcions predeterminades

- S'eliminen totes les funcions predeterminades de la Pràctica 1 llevat de `div`:
  - Les funcions `length`, `size` i `transpose` s'eliminen ja que el seu tractament requeriria poder tractar sobrecàrrega de funcions. També convé eliminar-les ja que no tenen correspondència amb les funcionalitats del nostre C3A.
  - Les funcions `zeros` i `ones` s'eliminen ja que caldria poder passar tipus de variables com a paràmetres de funcions.

- Encara que s'eliminin aquestes funcions, això no ha de significar cap problema. Per exemple, el llenguatge C no disposa de cap d'aquestes utilitats, i no obstant és un llenguatge que ho permet fer tot.
- L'especificació completa de la funció `div` és la següent:

  ```
  function div(i::Int32, j::Int32)::Int32
  ```
- No s'han de confondre les funcions predeterminades del codi font amb les funcions predeterminades del C3A. El C3A proporciona unes poques funcions predeterminades per a l'entrada i sortida de tipus bàsics.

Programa

- La sintaxi d'un programa és:

  llista_de_procediments_i_funcions
  llista_de_sentències
- L'exemple que hi ha a la secció de procediments i funcions té l'estructura d'un programa complet.
- Aquesta estructura fa que la seva implementació requereixi dues taules de símbols (TS) actives simultànies. La primera taula de símbols (TS1) correspon a tot el programa, i està activa durant tot el procés de compilació. La segona (TS2) es necessita per a compilar cadascun dels procediments i funcions. Aquesta TS2 pot ser una única taula de símbols que es buida quan s'acaba de compilar cada subprograma; alternativament, es pot crear una nova TS2 per a cada subprograma.
- La necessitat de tenir la TS1 preparada des de l'inici és perquè el programa principal (la llista de sentències després de la llista de subprogrames) necessita saber quins procediments i funcions s'han declarat abans, i quins són els seus patrons de paràmetres formals i de valor de retorn. És dir, s'ha de guardar a TS1 el nom de cada subprograma, i associat a cadascun d'ells, la informació del número de paràmetres formals i els seus tipus, així com el valor de retorn en el cas de funcions.

Codi de tres adreces

En el document *C3A.pdf* es descriu el llenguatge intermedi de codi de tres adreces de forma completament general. Per fer aquesta pràctica no es necessita tot, sinó només un subconjunt d'aquest llenguatge. Alguns detalls a tenir en compte són:

- Com no hi ha àmbits encaixats, totes les variables es poden considerar com a variables locals. En altres paraules, no cal registrar informació del nombre d'enllaços d'accés que cal seguir per a trobar les variables no locals, tal com es descriu a la documentació del C3A.
- Hi ha operadors del codi font que no tenen equivalent en C3A, per exemple, la potència o la divisió (`/`) d'enters. Per tant, és convenient ampliar la llibreria de funcions predeterminades de C3A amb unes quantes implementades per vosaltres, i que formarien part del vostre arxiu de sortida que conté el C3A.
- Com el llenguatge font no conté apuntadors, no cal utilitzar les proposicions de codi de tres adreces corresponents.
- La llista de sentències que formen el cos del programa (les que no estan dins de cap subprograma), podeu suposar que generen un procediment `Main` en C3A.

- Les expressions aritmètiques que només involucren literals s'haurien d'avaluar en memòria en temps de compilació, sense generar C3A pel seu càlcul. Per exemple: `2 + 3 * 4` és equivalent a `14`, i es pot calcular el seu valor en temps de compilació
- Les sentències que són només expressions (és dir, sense assignació) indiquen que es vol imprimir el seu valor per pantalla. Per a fer-ho, s'han d'utilitzar els procediments `PUT` de C3A predefinits.
- Per a poder imprimir per pantalla vectors i matrius, caldrà que definiu funcions en C3A que facin aquesta tasca, i així es simplificarà la vostra generació de codi intermedi C3A. Aquestes funcions també passarien a formar part de la vostra llibreria de funcions predeterminades de C3A.

Entrades i sortides

Entrada

- 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.

Sortida principal

- La sortida s'ha de posar en un arxiu de text amb sufix "-c3a.txt" (*nom-c3a.txt*), i també s'ha d'enviar a la consola (*stdout*). Ha de ser un programa en codi de tres adreces (C3A), respectant la descripció del C3A que es troba en el document *C3A.pdf*.
- L'arxiu de sortida contindrà per separat el C3A de cada procediment i funció que formen el vostre programa font, així com el codi corresponent al programa principal.
- La vostra llibreria de funcions predeterminades C3A la podeu posar en el mateix arxiu de sortida, o millor en un arxiu separat, per exemple, *llibreria-c3a.txt*.

Altres sortides

- 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 C3A amb 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.

Opcions

Enters i reals de diferents mides, cadenes, caràcters, operacions aritmètiques de vectors i matrius, registres, llistes, apuntadors, àmbits encaixats, etc. No s'admeten com a opcions els booleans ni sentències de flux de control diferents a les descrites.

Exemple

Entrada

```
function swap(v::Vector{Int32})
  temp = v[1]
  v[1] = v[2]
  v[2] = temp
end

function mean(a::Float64, b::Float64)::Float64
  return (a + b) / 2
end

v = [1; 2; 3; 4]
swap(v)
x = 10.0 + 7.0
i = 1
z = x + mean(x - 7, v[3 + i] * 3.5)
```

Sortida

```
 1:  START swap
 2:  $t01 := v[0]     # despl = (1-1)*4 = 0 bytes
 3:  temp := $t01
 4:  $t02 := v[4]     # despl = (2-1)*4 = 4 bytes
 5:  v[0] := $t02
 6:  v[4] := temp
 7:  RETURN
 8:  END

 1:  START mean
 2:  $t01 := a ADDD b         # suma de doubles
 3:  $t02 := $t01 DIVD 2.0    # divisió de doubles
 6:  RETURN $t02
 7:  END

 1:  START main
 2:  v[0] := 1        # despl = (1-1)*4 = 0 bytes
 3:  v[4] := 2        # despl = (2-1)*4 = 4 bytes
 4:  v[8] := 3        # despl = (3-1)*4 = 8 bytes
 5:  v[12] := 4       # despl = (4-1)*4 = 12 bytes
 6:  PARAM v
 7:  CALL swap,1      # crida a swap amb 1 param
 8:  x := 17.0        # càlcul 10.0+7.0 en compilació
 9:  i := 1
10:  $t01 := x SUBD 7.0
11:  $t02 := 3 ADDI i       # càlcul despl v[3 + i]
12:  $t03 := $t02 SUBI 1    #
13:  $t04 := $t03 MULI 4    #
14:  $t05 := v[$t04]
15:  $t06 := I2D $t05       # conversió tipus I -> D
16:  $t07 := $t06 MULD 3.5
17:  PARAM $t01
18:  PARAM $t07
19:  $t08 := CALL mean,2    # crida mean amb 2 params.
20:  $t09 := x ADDD $t08
21:  z := $t09
22:  HALT
23:  END
```

Execució del compilador

- 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-c3a.txt`
  - `$ ./nom_compilador nom`
  - `$ ./nom_compilador -i nom.jl -o nom-c3a.txt`

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:
  - `PR2-Nom1Cognom1-Nom2Cognom2.zip`.