Compiladors (CL)
GEI

Codi de Tres Adreces (versió 30-10-2021)

Esquema general

- Cada proposició de tres adreces ocupa una línia.
- Totes les línies de codi de tres adreces estan numerades consecutivament.
- Cada subprograma té associat una llista de proposicions de tres adreces numerades començant per 1.
- Pot haver-hi línies no numerades en blanc o que continguin únicament un comentari.
- Els comentaris comencen pel caràcter # i arriben fins el final de línia.
- Entre subprogrames diferents hi ha d'haver, com a mínim, una línia en blanc.

Elements bàsics

En les proposicions de tres adreces apareixen literals, variables, subprogrames i números de línia. Les regles per al seu tractament són les següents:

Tipus de dades

- A nivell de codi de tres adreces, totes les dades es tracten com a seqüències de bytes, sense tipus definits
- No obstant, existeixen tres tipus numèrics bàsics pels quals hi ha diversos operadors específics:

`Character`	(abreviatura: `C`; mida: 1 bytes)
`Integer`	(abreviatura: `I`; mida: 4 bytes)
`Long`	(abreviatura: `L`; mida: 8 bytes)
`Float`	(abreviatura: `F`; mida: 4 bytes)
`Double`	(abreviatura: `D`; mida: 8 bytes)

- No hi ha cap mena de comprovació de tipus, ja que no existeixen; l'únic que cal per a poder interpretar una dada (e.g., com a `Float`) és que la mida sigui la correcta (e.g., 4 bytes).

Literals

- Enters: sense punt decimal, per exemple: `123, 0, -456`
- Reals: amb punt decimal i notació científica o decimal, com: `123.2, 0.0, -1.23e-4`
- Caràcters: entre cometes simples i utilitzant la notació de C, com: `'A', '\n', '\x3F'`
- No hi ha literals de tipus Cadena; s'han de tractar caràcter a caràcter
- La mida en bytes dels literals depèn de les proposicions i/o operadors en què apareguin.

Variables

- Fan referència a noms existents dins de la taula de símbols, pels quals existeix espai reservat en els corresponents registres d'activació. Poden ser variables locals, no locals o temporals. Les variables temporals són sempre locals.
- Una ocurrència d'una variable en una proposició de tres adreces pot significar tant el valor emmagatzemat en el registre d'activació (valor de costat dret) com la seva adreça (valor de costat esquerra). Aquesta distinció depèn únicament del significat de la proposició.
- El nom de les variables temporals sempre comença per `$t` i van seguides d'un número, per exemple `$t05`.
- Per a variables locals, el nom en les proposicions de tres adreces coincideix amb el nom utilitzat a la taula de símbols.
- Per a variables no locals, el nom en les proposicions de tres adreces és el nom a la taula de símbols concatenat amb el nombre d'enllaços d'accés que cal seguir per a trobar el registre d'activació corresponent (aquest número es posa entre parèntesis).
- *Exemple*: si `loc` és una variable local, `$t05` és una variable temporal, i `nonloc` és una variable no local per a la què cal seguir 2 enllaços d'accés, totes de tipus enter, una possible proposició de tres adreces podria ser `$t05 := nonloc(2) ADDI loc`. Si `nonloc` fos una taula d'enters, podríem tenir `nonloc(2)[$t005] := loc`.

Subprogrames

- Fan referència a noms de procediments i funcions i, depenent de la implementació, també podrien fer referència a noms assignats dinàmicament pel compilador a blocs d'instruccions.
- Cada subprograma té associats: un nom a la taula de símbols de l'àmbit on s'ha declarat, un nom únic que s'utilitza en les proposicions de tres adreces de crida a subprogrames, una taula de símbols, un tipus de registre d'activació, i una seqüència de proposicions de tres adreces (el codi de tres adreces que es vol generar).
- El nom únic d'un subprograma és necessari per a distingir entre subprogrames que tinguin el mateix nom, ja sigui amb o sense sobrecàrrega (podria haver subprogrames del mateix nom encaixats en altres subprogrames, de manera que mai fossin visibles dos o més alhora).

Números de línia

- Les instruccions de salt requereixen que s'indiqui el número de línia de la proposició de tres adreces a la qual va dirigit el salt, per tant el millor és numerar-les totes.
- No s'admeten etiquetes simbòliques.
- En la generació de les proposicions de tres adreces corresponents a cada subprograma, s'han de numerar per ordre totes les línies, començant sempre per 1.

Proposicions de Tres Adreces

Inici de subprograma: **`START name`**

- Nom del subprograma: `name` és el nom del subprograma en codi de tres adreces, que pot no coincidir amb el nom que se li ha donat al subprograma en llenguatge font
- Restricció: tots els subprogrames que formen un programa han de tenir noms diferents en codi de tres adreces, tot i que en llenguatge font hi pugui haver repeticions (per exemple si hi ha sobrecàrrega, o entre subprogrames del mateix nom que es troben en àmbits que no són visibles entre sí); per aquesta raó direm que `name` és el nom únic del subprograma font original
- Observació: Si el llenguatge font no admet sobrecàrrega ni encaixament de subprogrames, aleshores el nom únic pot coincidir amb el nom del subprograma
- *Exemple*: un procediment

  ```
  procedure P(I: Integer; F: Float)
  ```

  podria donar lloc a un inici de subprograma del tipus

  ```
  START P_03
  ```

  on `p_03` és el nom únic que se li ha assignat al procediment `p`

Final de subprograma: **`END`**

Assignació amb operador unari: **`x := op y`**

- Operands: `x` ha de ser una variable, `y` una variable o un literal
- Operadors de canvi de signe d'enters i reals:

  ```
  op = "CHSL", "CHSI", "CHSD", "CHSF"
  ```
- Operadors de conversió de tipus enters:

  ```
  op = "L2I", "I2L"
  ```
- Operadors de conversió de tipus reals:

  ```
  op = "F2D", "D2F"
  ```
- Operadors de conversió entre enters i reals:

  ```
  op = "L2D", "L2F", "D2L", "F2L"
  op = "I2D", "I2F", "D2I", "F2I"
  ```
- Operadors de conversió entre tipus enters i caràcters:

  ```
  op = "L2C", "I2C", "C2L", "C2I
  ```
- No hi ha més operadors unaris
- Significats: `CHS`=canvi de signe, `L`=Long, `I`=Integer, `C`=Character, `D`=Double, `F`=Float
- Observació: en la conversió entre enters i caràcters, l'enter representa el codi ASCII
- *Exemple*: `x := I2F i`

Assignació amb operador binari: **`x := y op z`**

- Operands: `x` ha de ser una variable, `y` i `z` variables o literals
- Operacions aritmètiques d'enters:

  ```
  op = "ADDL", "SUBL", "MULL", "DIVL", "MODL"
  op = "ADDI", "SUBI", "MULI", "DIVI", "MODI"
  ```
- Operacions aritmètiques de reals:

  ```
  op = "ADDD", "SUBD", "MULD", "DIVD"
  op = "ADDF", "SUBF", "MULF", "DIVF"
  ```
- No hi ha més operadors binaris; els operadors binaris relacionals que s'expliquen més avall (e.g. `"EQ"` i `"GTI"`) no es poden utilitzar en aquest tipus d'assignació
- Significats: `ADD`=suma, `SUB`=resta, `MUL`=multiplicació, `DIV`=divisió, `MOD`=mòdul
- *Exemple*: `x := y MULF z`

Còpia: **`x := y`**

- Operands: `x` ha de ser una variable, `y` una variable o un literal
- Restricció: `x` i `y` han de ser objectes de la mateixa mida
- Significat: es fa una còpia de tots els bytes de `y` cap a `x`
- *Exemple*: amb el codi font següent

```
A: array(1..6) of Double;
type Taula is array(1..6) of Double;
T: Taula := Taula(A);  -- Conversió explícita de tipus
                       -- estructualment equivalents
Pi: Float := 3.141592654;
```

quedaria el següent codi de tres adreces

```
T := A                # còpia de 48 bytes
Pi := 3.141592654     # còpia de 4 bytes
```

Salt incondicional: **`GOTO L`**

- L'etiqueta L no pot ser simbòlica, ha d'indicar el número de línia d'una proposició de tres adreces
- *Exemple*: `GOTO 100`

Salt condicional: **`IF x oprel y GOTO L`**

- L'etiqueta L no pot ser simbòlica, ha d'indicar el número de línia d'una proposició de tres adreces
- Operands: `x` i `y` poden ser variables o literals
- Restricció: `x` i `y` han de ser objectes del mateix tamany
- Igualtat i desigualtat:

```
oprel = "EQ", "NE"
```

- Comparació d'enters:

```
oprel = "LTL", "LEL", "GTL", "GEL"
oprel = "LTI", "LEI", "GTI", "GEI"
oprel = "LTC", "LEC", "GTC", "GEC"
```

- Comparació de reals:

```
oprel = "LTD", "LED", "GTD", "GED"
oprel = "LTF", "LEF", "GTF", "GEF"
```

- Significats: `EQ`=igual, `NE`=diferent, `LT`=més petit, `LE`=més petit o igual, `GT`=més gran, `GE`=més gran o igual
- *Exemple*: `IF i LEI j GOTO 100`

Consulta desplaçada: **`x := y[i]`**

- Operands: `x` i `y` han de ser variables, `i` un desplaçament (mesurat en bytes) respecte la posició inicial de `y`
- Significat: es fa una còpia de tants bytes com té `x` des de l'adreça `y+i` cap a `x`
- Restricció: la mida de `y` ha de ser més gran o igual a `i` més la mida de `x`
- *Exemple*: si tenim definits els tipus

```
type Taula is array(5..10) of Double;
type Complex is record
  Voltes: Integer;
  Re, Im: Double;
end record;
```

i les variables

```
I, J: Integer;
X: Double;
T: Taula;
C: Complex;
```

l'assignació

```
X := T(7) + C.Im;
```

genera codi del tipus

```
$t1 := T[16]          # desplaçament 16 és (7 - 5) * 8 bytes
$t2 := C[12]          # desplaçament 12 és 4 + 8 bytes
$t3 := $t1 ADDD $t2
X := $t3
```

Assignació desplaçada: `x[i] := y`

- Operands: `x` ha de ser una variable, `y` una variable o un literal, `i` un desplaçament (mesurat en bytes) respecte la posició inicial de `x`
- Significat: es fa una còpia de tots els bytes de `y` cap a l'adreça `x+i`
- Restricció: la mida de `x` ha de ser més gran o igual a `i` més la mida de `y`
- *Exemple*: utilitzant els tipus i variables de l'exemple anterior, les assignacions

```
C.Re := T(I);
T(J) := T(2 * I)
```

generen codi del tipus

```
$t1 := I SUBI 5          # desplaçament de T(I)
$t2 := $t1 MULI 8        #
$t3 := T[$t2]
C[4] := $t3              # final primera assignació
$t4 := 2 MULI I          # desplaçament de T(2 * I)
$t5 := $t4 SUBI 5        #
$t6 := $t5 MULI 8        #
$t7 := T[$t6]
$t8 := J SUBI 5          # desplaçament de T(J)
$t9 := $t8 MULI 8        #
T[$t9] := $t7            # final segona assignació
```

- Observació: és incorrecte que en una mateixa instrucció hi hagi al mateix temps una assignació i una consulta desplaçada, e.g. `T[$t9] := T[$t6]`, ja que això constituiria codi de quatre adreces!

Procediments

- Paràmetre: **`PARAM x`**
- Crida d'un procediment: **`CALL p,n`**
- Significat: `p` és el nom únic del procediment cridat, i `n` el seu nombre d'arguments
- Retorn d'un procediment: **`RETURN`**
- Procediments predefinits de sortida a pantalla (un únic argument, l'objecte que es vol fer sortir per pantalla): `p = "PUTL", "PUTI", "PUTC", "PUTD", "PUTF"`
- Significats: `PUT`=escriure
- Sintaxi: totes les instruccions `PARAM` que corresponen a una crida han d'estar immediatament abans de la crida a `CALL`, és dir, no hi pot haver altres proposicions intercalades entre els `PARAM` o entre el darrer `PARAM` i el `CALL`.
- *Exemple*: una crida

```
P(A + B, C * D)
```

al procediment

```
procedure P(I: Integer; X: Float)
```

genera codi del tipus

```
$t1 := A ADDI B            # càlcul de "tots" els paràmetres actuals
$t2 := C MULF D            # abans de la crida
PARAM $t1
PARAM $t2
CALL P_03,2
```

- *Exemple*: un procediment

```
procedure Swap(T: Taula; I, J: Integer) is
  Temp: Float;
begin
  Temp := T(I);
  T(I) := T(J);
  T(J) := Temp;
end Swap
```

genera codi del tipus

```
 1:  START Swap_01          # inici procediment
 2:  $t01 := I SUBI 5       # càlcul desplaçament T(I)
 3:  $t02 := $t01 MULI 8    #
 4:  $t03 := T[$t02]
 5:  Temp := $t03
 6:  $t04 := J SUBI 5       # càlcul desplaçament T(J)
 7:  $t05 := $t04 MULI 8    #
 8:  $t06 := T[$t05]
 9:  $t07 := I SUBI 5       # càlcul desplaçament T(I)
10:  $t08 := $t07 MULI 8    #
11:  T[$t08] := $t06
12:  $t09 := J SUBI 5       # càlcul desplaçament T(J)
13:  $t10 := $t09 MULI 8    #
14:  T[$t10] := Temp
15:  RETURN                 # retorn procediment
16:  END                    # final procediment
```

Funcions

- Paràmetre: **`PARAM x`**
- Crida d'una funció: **`z := CALL f,n`**
- Significat: `f` és el nom únic de la funció cridada, i `n` el seu nombre d'arguments
- Retorn d'una funció: **`RETURN x`**
- Funcions predefinides de lectura de teclat (sense arguments, i retornen el valor que s'ha introduït per teclat): `f = "GETL", "GETI", "GETC", "GETD", "GETF"`
- Funció predefinida d'adquisició dinàmica de memòria (un argument enter positiu, el nombre de bytes contigus que es volen reservar, a l'estil del `malloc` del llenguatge C, i retorna un apuntador al bloc reservat, és dir, l'adreça del seu primer byte): `p = "ALLOC"`
- Significats: `GET`=llegir, `ALLOC`=reservar
- Sintaxi: totes les instruccions `PARAM` que corresponen a una crida han d'estar immediatament abans de la crida a `CALL`, és dir, no hi pot haver altres proposicions intercalades entre els `PARAM` o entre el darrer `PARAM` i el `CALL`.
- *Exemple*: una crida

```
X := 2 + 3 * Get_Integer;
```

a la funció predefinida

```
function Get_Integer return Integer
```

genera codi del tipus

```
$t1 := call GETI,0
$t2 := 3 MULI $t1
$t3 := 2 ADDI $t2
X := $t3
```

- *Exemple*: una funció

```
function Sum(I: Integer; X: Float) is
begin
  return Float(I) + X;
end Swap
```

genera codi del tipus

```
1:  START Sum_01              # inici funció
2:  $t01 := I2F I             # conversió de tipus I -> F
3:  $t02 := $t01 ADDF X
4:  RETURN $t02               # retorn funció
5:  END                       # final funció
```

Consulta amb desreferència: **`x := *y`**

- Operands: `x` i `y` han de ser variables, `y` un apuntador
- Significat: es fa una còpia de tants bytes com té `x` des de l'adreça apuntada per `y` cap a `x`

Assignació amb desreferència: **`*x := y`**

- Operands: `x` ha de ser variable i apuntador, `y` una variable o un literal
- Significat: es fa una còpia de tots els bytes de `y` cap a l'adreça apuntada per `x`

Obtenció d'una referència: **`x := &y`**

- Operands: `x` i `y` han de ser variables, `x` és un apuntador
- Significat: es fa una còpia de l'adreça de `y` en `x`

Finalització de l'execució: **`HALT`**