Arrays und Array Slicing

Mit Arrays arbeiten, leicht gemacht.

Mark Zeman, Philip Stark

FHNW - Compilerbau

18.11.2014

- `http://github.com/hellerbarde/cpib`

Übersicht

Sprachdesign

Technisches

Beispiel

Arrays

Wir definieren Arrays so:

- Multidimensional
- Primitive Typen

Die Deklaration sieht mittlerweile so aus:

```
var a:array (5) int;
```

und der Zugriff so:

```
a[5];
```

Array Slicing

Was?

- Extraktion eines Teilarrays

```
a := [1,2,3,4,5];

print(a[0..3]);

--------------------
output => [1,2,3,4]
```

Array Slicing

Warum?

- Selten in Sprachen umgesetzt
- Expressive Syntax
- Übersichtlicher Array-Handling Code

Ein Slice hat die folgende Syntax:

```
a[2..4];
```

Gedanken zur Deklaration

Mehrere Versuche, bis wir zur definitiven Syntax kamen:

- `var a:TYPE[LENGTH][DIMENSIONS];`
- `var b:array (array or type) LENGTH;`
- `var c:array LENGTH (array or type);`

Schlussendlich:

- `var d:array (LENGTHS) type;`

Gedanken zu Initialisierung und Zugriff

- Immer vollständig initialisiert
- Syntaktischer Zucker um Arrays zu “nullen”
- Kann genutzt werden um beliebig zu füllen

Die Syntax für die Initialisierung und den Zugriff sieht so aus:

- `a init := [0,1,2,3]`
- `b init := fill 5`
- `c init := a[3]`

Gedanken zur Syntax der Array-Slices

- `b := a[2..4]`
- Ähnlichkeit zum Zugriff
- Kein Überladen des Minus Operators

Lexikalische Erweiterungen

- Nur weniges nötig
- `FILL` für den syntaktischen Zucker
- `DOTDOT` damit wir Minus nicht überladen

Pattern	Token
`[`	LBRACKET
`]`	RBRACKET
`fill`	FILL
`array (<int>{,<int>}) <type>`	(ARRAY,Length,Type)
`..`	DOTDOT

Grammatikalische Produktionen

```
cmd      ::= SKIP
           | expr ':=' [FILL] expr
           [...]
typedIdent ::= IDENT ':'(ATOMTYPE
                 | ARRAY '(' expr {',' expr} ')' TYPE)
factor ::= LITERAL
         | arrayLiteral
         | IDENT [INIT | exprList | arrayIndex]
         [...]

arrayIndex::= '[' expr ['..' expr] ']' {arrayIndex}
arrayLiteral ::= '[' arrayContent ']'
arrayContent ::= LITERAL {',' LITERAL}
               | arrayLiteral {',' arrayLiteral}
```

Matrix Multiplikation

```
do
  a init := [[1,2,3],[4,5,6]];
  b init := [[1,2],[3,4],[5,6]];
  c init := fill 0;
  [skip restliche Initialisierungen]
  while i > 2 do
    while j > 2 do
      while k > 3 do
        c[i][j] := c[i][j] + a[i][k] * b[k][j];
        k := k+1
      endwhile
      j := j+1
    endwhile
    i := i+1
  endwhile
endprogram
```

Datensatzauswertung

- Wetterstation
- Datum, Temperatur [°C], Niederschlag [mm]
- Output: Anzahl Tage mit `T > 25 && N > 5`

```
global
  var datapoint : array (3) int;
do
  while i > 100 do
    datapoint := input[i*3 .. i*3 + 2]
    if datapoint[1] > 25 && datapoint[2] > 5 then
      result := result + 1;
    else
    endif
    i := i+1
  endwhile
  return result;
endfun
```