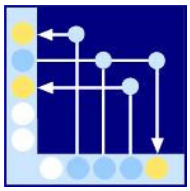
Fakultät Elektronik und Informatik
Studienbereich Informatik



Compilerbau-Praktikum

Lehrveranstaltung im Wintersemester 2022/2023
Prof. Dr. habil. Christian Heinlein

9. Aufgabenblatt (13. Dezember 2022)

Aufgabe 9: Ausschlussdeklarationen

Definieren Sie einen vordefinierten Operator `excl•end`, der zum Beispiel wie folgt verwendet werden kann, um Ausschlussangaben für benutzerdefinierte Operatoren zu definieren:

```
(x) "²" -> (x * x);

excl (print 1)²; (1 + 2)²; (1 * 2)² end;

print 10²;
print 3 + 4 * 5²
```

Die Werte `1` und `2` sind hierbei willkürlich und stehen jeweils stellvertretend für einen beliebigen Teilausdruck.

Der gewünschte Effekt solcher Deklarationen kann wie folgt erreicht werden:

- Der Operator `excl•end` besitzt eine geeignete Prolog- oder Epilogfunktion, die seinen Operanden rekursiv durchläuft.
- Wenn ein Teilausdruck dieses Operanden eine Anwendung des vordefinierten Klammeroperators ist, erhält der Parameter, zu dem dieser Teilausdruck gehört, eine zusätzliche Ausschlussangabe für den Operator, der sich an der Spitze des geklammerten Ausdrucks befindet.

  Im obigen Beispiel erhält der Parameter `x` des Operators `•²` somit nacheinander Ausschlussangaben für die Operatoren `print•`, `•+•` und `•*•`.

  Die Ausschlussangaben erhalten je nach Position des Parameters in der Signatur seines Operators die Kennzeichnungen `FR` (vorderer Operand), `MLR` (mittlerer Operand) oder `BL` (hinterer Operand).

Der Resultatwert eines Ausdrucks `excl ... end` ist nil.

Aufgabe 10: Statische Operatoren

Erweitern Sie die bis jetzt entwickelte Programmiersprache um statische Operatoren, die sich nur in folgenden Punkten von anderen benutzerdefinierten Operatoren (die zur Unterscheidung auch als dynamische Operatoren bezeichnet werden) unterscheiden:

- Bei der Deklaration eines statischen Operators wird anstelle des Pfeils `->` ein Doppelpfeil `=>` verwendet.
- Die Auswertungsfunktion eines statischen Operators überprüft nach der Auswertung der Operanden, ob der Operator schon einmal auf die gleichen Operandenwerte angewandt wurde. Wenn ja, wird der gleiche Wert wie da-

mals zurückgeliefert, und die Implementierung des Operators wird nicht ausgewertet. Andernfalls wird wie bei einem dynamischen Operator die Implementierung ausgewertet und der resultierende Wert zurückgeliefert.

Um das beschriebene Laufzeitverhalten zu realisieren, muss es zu jeder Ausprägung eines statischen Operators zur Laufzeit eine Tabelle zur Speicherung seiner Rückgabewerte geben, die im tabellenwertigen Attribut

```
ATTRT(mem_, Value, seq<Value>, Value)
```

des Objekts gespeichert werden kann, das von der Auswertungsfunktion des Operatordeklarationsoperators geliefert wurde.[1] Als Tabellenschlüssel wird die Sequenz aller Operandenwerte verwendet.

Damit das tabellenwertige Attribut wie gewünscht funktioniert, muss für den Typ `seq<Value>` folgendes definiert werden:

- Ein Gleichheitsoperator

  ```
  bool operator== (const seq<Value>& vs1, const seq<Value>& vs2) { ..... }
  ```

  der genau dann `true` liefert, wenn die Sequenzen `vs1` und `vs2` paarweise gleiche Werte enthalten, wobei die Gleichheit zweier Werte wie beim Vergleichsoperator = der Programmiersprache definiert ist.

- Eine Spezialisierung von `std::hash` mit einer Elementfunktion `operator()`, die exakt wie folgt definiert ist und die den Streuwert der Sequenz `vs` auf irgendeine sinnvolle Art und Weise berechnet:

  ```
  template <>
  struct std::hash<seq<Value>> {
      size_t operator() (const seq<Value>& vs) const { ..... }
  };
  ```

  Hinweis: Wenn `Value v` ein synthetischer Wert ist, kann seine ID `v.id` mit Typ `char*` verwendet und in `size_t` umgewandelt werden.

Aufgabe 11: Anwendungsprogramme

Teilaufgabe 11.a)

Definieren Sie den Operator `ack` • • aus Aufgabe 8b als statischen statt als dynamischen Operator und vergleichen Sie die Laufzeiten für unterschiedliche Parameterwerte!

Definieren Sie einen Operator •`over`• zur rekursiven Berechnung von Binomialkoeffizienten gemäß folgender Definition: $\binom{n}{k} = \begin{cases} 1 & \text{für } k = 0 \text{ oder } k = n \\ \binom{n-1}{k-1} + \binom{n-1}{k} & \text{sonst} \end{cases}$

Vergleichen Sie wiederum die Laufzeiten, wenn der Operator dynamisch oder statisch ist!

[1] Wenn ein statischer Operator in der Implementierung eines anderen Operators definiert wird, gibt es zur Laufzeit bei jeder Auswertung dieser Implementierung eine neue Ausprägung des statischen Operators. Deshalb darf die Tabelle nicht beim Operator (Typ `Oper`) gespeichert werden, sondern bei dem zuvor genannten Laufzeitwert (Typ `Value`).

Teilaufgabe 11.b)

Statische Operatoren können zum Beispiel wie folgt zur Realisierung von Datenstrukturen verwendet werden:

```
(obj) "." (attr) => (uniq);
(obj) "?" (attr) -> (?(obj.attr));
excl ... end;

x := uniq;
y := uniq;

p1 := uniq;
p2 := uniq;
p1.x =! 3;
p1.y =! 4;
p2.x =! 5;

print p1?x;
print p1?y;
print p2?x;
print p2?y
```

Erläuterungen:

- *obj*`.`*attr* liefert für jede Kombination eines „Objekts“ *obj* und eines „Attributs“ *attr* einen eindeutigen synthetischen Wert, der als Variable verwendet werden kann.
- *obj*`?`*attr* liefert den Wert der entsprechenden Variablen.
- Damit weist *obj*`.`*attr* `=!` *val* der entsprechenden Variablen den Wert *val* zu, der anschließend mit *obj*`?`*attr* abgefragt werden kann.
- Sowohl Objekte als auch Attribute können jeweils als Konstanten mit eindeutigem synthetischem Wert definiert werden.
- Damit die obigen Ausdrücke eindeutig sind, müssen entsprechende Ausschlussdeklarationen angegeben werden.

Anstelle von Attributen können auch Zahlenwerte verwendet werden, um Containerobjekte ähnlich wie Arrays zu realisieren, zum Beispiel:

```
squares := uniq;

i := uniq;
i =! 1;
while ?i <= 10 do
    squares.?i =! ?i^2;
    i+!
end;

print squares?5
```

Implementieren Sie auf diese Weise einen Operator, der ein Array mit einem Sortierverfahren Ihrer Wahl sortiert!