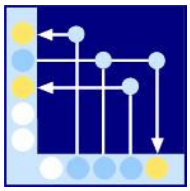
Hochschule Aalen
Fakultät Elektronik und Informatik
Studienbereich Informatik



Compilerbau-Praktikum

Lehrveranstaltung im Sommersemester 2021
Prof. Dr. habil. Christian Heinlein

2. Aufgabenblatt (29. März 2021)

Aufgabe 2: C++-Bibliothek *lib*$_{\texttt{CH}}$: Offene Typen

Arbeiten Sie sich anhand der Dokumentation zur Bibliothek *lib*$_{\texttt{CH}}$ in das Thema offene Typen ein! Wichtig sind zunächst vor allem die Teilabschnitte 6.1.1 bis 6.1.4 sowie 6.1.10. Für den letzten Teil von Teilaufgabe b ist außerdem Teilabschnitt 6.1.7 wichtig.

Implementieren Sie mit dem erworbenen Wissen u. a. folgendes:

a) Der offene Typ `Expr` repräsentiert arithmetische Ausdrücke `x` unterschiedlicher Art, konkret konstante Ausdrücke mit Wert `x(value)`, Variablen mit Namen `x(name)`, unäre Ausdrücke mit Operator `x(oper)` (entweder `"+"` oder `"-"`) und Operand `x(body)` sowie binäre Ausdrücke mit Operator `x(oper)` (`"+"`, `"-"`, `"*"` oder `"/"`) und Operanden `x(left)` und `x(right)`.
Der Typ des Attributs `value` soll `double` sein, der Typ der Attribute `name` und `oper` soll `str` sein, und der Typ der Attribute `body`, `left` und `right` soll wiederum `Expr` sein.

   Für einen `double`-Wert `v` liefert `expr(v)` einen konstanten Ausdruck mit Wert `v`.
Für einen Variablennamen `n` liefert `expr(n)` eine Variable mit dem Namen `n`.
Für einen Operatornamen `op` und einen Ausdruck `x` liefert `expr(op, x)` einen unären Ausdruck mit Operator `op` und Operand `x`.
Für einen Operatornamen `op` und Ausdrücke `x` und `y` liefert `expr(x, op, y)` einen binären Ausdruck mit Operator `op` und Operanden `x` und `y`.

   Für einen Ausdruck `x` berechnet `eval(x)` den Wert des Ausdrucks und liefert ihn zurück.
Der Wert einer Variablen wird hierfür aus einer globalen Tabelle mit Typ `std::unordered_map<str, double>` ermittelt.

   Für einen Ausdruck `x` liefert `fmt(x)` eine Zeichenkettendarstellung des Ausdrucks, z. B. `(1+(a*2))`.

b) Der offene Typ `Person` repräsentiert Personen `p` mit beliebig vielen Vornamen `p(firstnames)` und Nachname `p(name)`, jeweils mit Typ `str`.

   Für eine Person `p` liefert `fmt(p, abbr)` eine Zeichenkettendarstellung der Person, die für `abbr` gleich `false` aus allen Vornamen und dem Nachnamen der Person, jeweils getrennt durch ein Leerzeichen, besteht. Für `abbr` gleich `true` wird jeder Vorname durch seinen ersten Buchstaben und einen Punkt ersetzt. Der Parameter `abbr` ist optional; wenn er fehlt, ist sein Wert `false`.

   Ergänzen Sie ein einwertiges Attribut `spouse_` mit Zieltyp `Person`, das nicht direkt verwendet werden soll, sowie ein virtuelles einwertiges Attribut `spouse` mit folgender Bedeutung:
Für Personen `p` und `q` setzt `p(spouse, q)` oder `q(spouse, p)` den Ehepartner `p(spouse)` von `p` auf `q` und den Ehepartner `q(spouse)` von `q` auf `p`.
Wenn `p` und/oder `q` bereits einen anderen Ehepartner `pp` bzw. `qq` besitzt, wird die Verbindung zu diesem gelöst,

d. h. `pp` und/oder `qq` hat anschließend keinen Ehepartner mehr.
`p` oder `q` kann auch nil sein, um lediglich den aktuellen Ehepartner von `q` bzw. `p` zu entfernen. (Wenn beide nil sind, sind die Operationen wirkungslos.)
Zum Beispiel:

```
Person p1 = true;
Person p2 = true;
Person q1 = true;
Person q2 = true;

p1(spouse, q1);          // p1 <-> q1
q2(spouse, p2);          // p2 <-> q2

p1(spouse, q2);          // p1 <-> q2, q1 und p2 haben keinen Ehepartner mehr
```

Aufgabe 3: Zentrale Datenstrukturen des MOSTflexiPL-Compilers

Arbeiten Sie sich anhand der Dokumentation in die zentralen Datenstrukturen des MOSTflexiPL-Compilers ein! (Teilabschnitt 1.1.6 ist momentan noch nicht wichtig.)

Folgende Typen und Operatoren seien bereits definiert:

```
#include "data.h"

// Metatyp type mit zugehörigem Operator.
Type type_type = true;
Oper type_oper = Oper(res, type_type)(sig, Sig(Part(name, "type")));
Type dummy = type_type(type, type_type)(oper, type_oper);

// Typ bool mit zugehörigem Operator.
Oper bool_oper = Oper(res, type_type)(sig, Sig(Part(name, "bool")));
Type bool_type = Type(type, type_type)(oper, bool_oper);

// Typ int mit zugehörigem Operator.
Oper int_oper = Oper(res, type_type)(sig, Sig(Part(name, "int")));
Type int_type = Type(type, type_type)(oper, int_oper);
```

a) „Übersetzen“ Sie mit dem erworbenen Wissen die folgenden MOSTflexiPL-Deklarationen in entsprechende `Oper`-Objekte (ohne Werte für die in der Dokumentation ausgegrauten Attribute) mit zugehörigen `Part`-Hilfsobjekten:

```
a : bool;
b : bool;
c : int;
d : int;
(int) "+" (int) -> (int)
print (int) [dec|oct|hex|bin] -> (bool)
sum of (int) { and (int) } end -> (int)
dnf (bool) { and (bool) } { or (bool) { and (bool) } } end -> (bool)
```

Zum Beispiel:

```
// Konstante a mit Typ bool.
Oper a = Oper(res, bool_type)(sig, Sig(Part(name, "a")));

// Anonyme Parameter p und q mit Typ int.
Par p = Par(res, int_type);
Par q = +p;

// Binärer Plus-Operator.
Oper plus = Oper(res, int_type)
    (sig, Sig(Part(par, p), Part(name, "+"), Part(par, q)));

// Variadischer sum-Operator.
Oper sum = ......
```

Definieren Sie sich bei Bedarf geeignete Hilfsobjekte und -funktionen zur Vereinfachung!

b) „Übersetzen“ Sie entsprechend die folgenden MOSTflexiPL-Ausdrücke in entsprechende `Expr`-Objekte (ohne Werte für die in der Dokumentation ausgegrauten Attribute) mit zugehörigen `Pass`- und `Item`-Hilfsobjekten, die bei Bedarf die zuvor erstellen `Oper`-Objekte verwenden:

```
c
c + d
print c
print d hex
sum of c and d end
sum of c and c + d and d end
sum of sum of c and d end and sum of d and c end end
dnf a or a and b or a and b and print c end
```

Zum Beispiel:

```
// Verwendung der Konstanten c und d.
Expr c_ = Expr(oper, c)(type, int_type);
Expr d_ = Expr(oper, d)(type, int_type);

// c + d
Expr c_plus_d = Expr(oper, plus)(type, int_type)
    (row, Row(Item(opnd, c_), Item(word, "+"), Item(opnd, d_)));

// sum of c and c + d and d end
Expr sum_of_c_and_c_plus_d_and_d_end =
    Expr(oper, sum)(type, int_type)(row, Row(
        Item(word, "sum"),
        Item(word, "of"),
        Item(opnd, c_),
        Item(passes,
            Pass(choice, A)(branches, Row(
                Item(word, "and"),
                Item(opnd, c_plus_d))),
            Pass(choice, A)(branches, Row(
                Item(word, "and"),
                Item(opnd, d_)))
        ),
        Item(word, "end")
    ));
```