Formal Semantics of Programming Languages
Exercise (Deadline: June 24)

Wolfgang Schreiner
RISC-Linz

May 19, 1999

Task

Design a small imperative programming language, develop its denotational semantics, and implement an interpreter for this language.

The language should allow to write programs like

```
const n = 10
var a[n]
var i
var s
...
while i < n do
  var b
  if a[i] > 0 then
    b = a[i]
  else
    b = -a[i]
  end
  s = s+b
end
```

i.e., have a nested block structure, scalar integer constants, scalar and array variables over the integers, assignments, sequences, conditionals and while loops. The program starts with an empty (undefined) input store.

First design the abstract syntax of the language and develop its denotational semantics. Then implement the semantic equations by an interpreter. The interpreter should print the result store after termination.

Deliverables

The deliverables of this exercise are a small report describing

1. Abstract syntax
2. Denotational semantics (including semantic algebras)
3. Commented source code of the interpreter
4. Output of some test runs
5. Comments on your design, correspondence/difference of semantics and implementation, etc.

The goal of implementing the interpreter is not efficiency but elegance and demonstrating the correspondence with the semantic definition.

Implementation Hints

Choose your favorite programming language (Mathematica, Lisp, ML, CafeObj, Java, C++, C, ...) in order to implement an interpreter that operates on *abstract syntax trees* (i.e., you need not parse *concrete* syntax) based on the semantic equations.

Abstract Syntax Trees

In Mathematica, such an abstract syntax tree can be represented by a term of the form

```
WhileStat[LessThan[Variable["i"], Number[10]],
  Block["b", CondStat[...]]]
```

while in Java (similar in C++) the representation can be an object

```
new WhileStat(new LessThan(new Variable("i"), new Number(10)),
  new Block(...))
```

with classes like

```
class WhileStat implements Statement
{
  public WhileStat(Expression e, Statement s)
  ...
}
```

In an purely imperative language (say C, not recommended), abstract syntax trees can be represented by pointers to structures, say

```
typedef struct{ Expression *exp; Statement *stat } WhileStat;
WhileStat *whileStat(Expression *exp, Statement *stat) {
  WhileStat *w = (WhileStat*)malloc(sizeof WhileStat);
  w->exp = exp;
  w->stat = stat;
  return w;
}
```

and constructed by function applications like

```
whileStat(lessThan(variable("i"), number(10)),
  block(...))
```

Correspondingly the store can be implemented in different ways in different languages, e.g. as a sequence of (location, value) pairs that is extended by every new assignment.

The constant/variable environment can be implemented, e.g., by a linked list of (name, value) pairs.

Evaluation

The interpreter should provide a function like

```
eval(program)
{
  Store resultStore = P(program, emptyStore);
  print(resultStore);
}
```

which interprets the denoted program and prints the result store after execution.