Programming Language Technology

Exam, 15 January 2016 at 14.00 – 18.00 in H

Course codes: Chalmers DAT151, GU DIT231. As re-exam, also DAT150, TIN321 and DIT229/230.
Teacher: Fredrik Lindblad (tel. 031-7721051)

Grading scale: Max = 60p, VG = 5 = 48p, 4 = 36p, G = 3 = 24p.
Allowed aid: an English dictionary.
Exam review: will be announced on `plt-2015-lp2` mailing list.

Please answer the questions in English. Questions requiring answers in code can be answered in any of: C, C++, Haskell, Java, or precise pseudocode.

For any of the six questions, an answer of roughly one page should be enough.

Question 1 (Grammars): Write a labelled BNF grammar that covers the following constructs in a C-like imperative language: A program is a list of statements. Types are `int` and `bool`. Statement constructs are:

- `while` loops
- variable declarations (e.g. `int x;`), not mutliple variables, no initial value
- expression statements (`E;`)

Expression constructs are:

- identifiers/variables
- integer literals
- less-than comparisons (`E < F`)
- assignments of identifiers (`x = E`)
- pre-increments of identifiers (`++x`)

Operator precedences and associativity should follow the C standard (less-than is left associative). You can use the standard BNFC categories `Integer` and `Ident` as well as list short-hands, and `terminator`, `separator` and `coercions` rules. (10p)

SOLUTION:

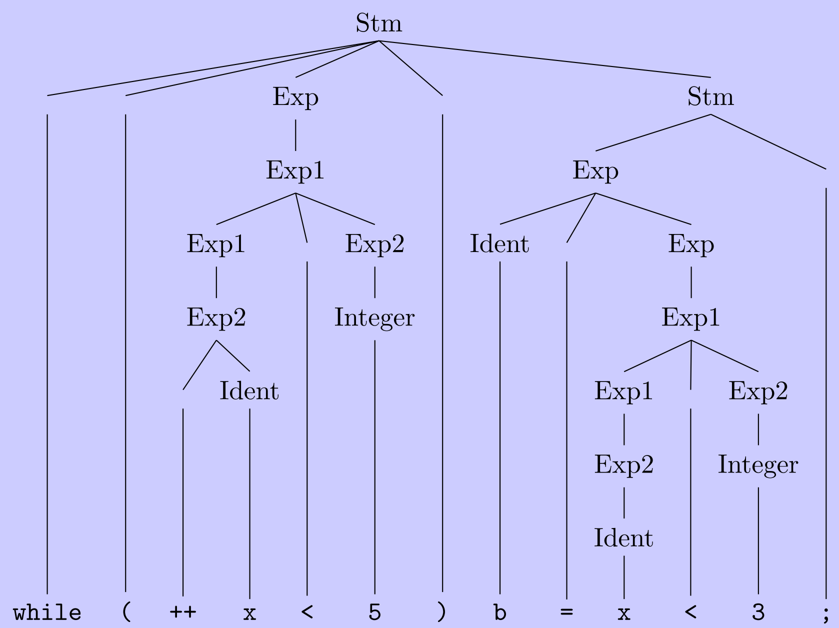
```
PStms.    Prg  ::= [Stm] ;

terminator Stm "" ;

TInt.     Type ::= "int" ;
TBool.    Type ::= "bool" ;

SWhile.   Stm  ::= "while" "(" Exp ")" Stm ;
SDecl.    Stm  ::= Type Ident ";" ;
SExp.     Stm  ::= Exp ";" ;

EId.      Exp2 ::= Ident ;
EInt.     Exp2 ::= Integer ;
EPreIncr. Exp2 ::= "++" Ident ;
ELt.      Exp1 ::= Exp1 "<" Exp2 ;
EAss.     Exp  ::= Ident "=" Exp ;

coercions Exp 2 ;
```

Question 2 (Trees): Show the parse tree and the abstract syntax tree of the statement

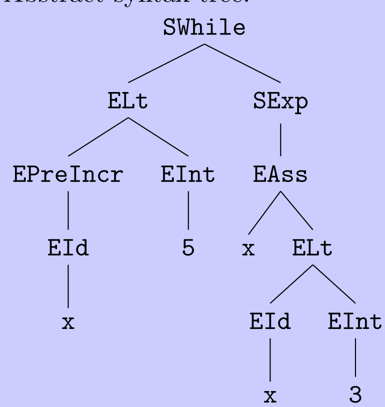
`while (++x < 5) b = x < 3;`

in the grammar that you wrote in question 1. (10p)

SOLUTION: Parse tree:



Abstract syntax tree:



Question 3 (Typing and evaluation):

A. Write typing rules or syntax-directed type-checking code (or pseudocode) for the *statement* constructs of Question 1. The variable context must be made explicit. Note that, due to the absence of statement blocks, a *stack* of contexts is not required. You can refer to the expression type-checking judgment or function without defining it. (5p)

SOLUTION:

$$\frac{\Gamma \vdash e : \texttt{bool} \qquad \Gamma \vdash s \ \mathit{valid}}{\Gamma \vdash \texttt{while (}e\texttt{) } s\texttt{;} \ \mathit{valid}}$$

$$\frac{\Gamma \vdash e : T}{\Gamma \vdash e\texttt{;} \ \mathit{valid}}$$

$$\frac{\Gamma, x : T \vdash s_2 \ldots s_n \ \mathit{valid}}{\Gamma \vdash T \ x\texttt{;} s_2 \ldots s_n \ \mathit{valid}} x \notin \Gamma$$

$$\frac{\Gamma \vdash s_1 \ \mathit{valid} \qquad \Gamma \vdash s_2 \ldots s_n \ \mathit{valid}}{\Gamma \vdash s_1 \ldots s_n \ \mathit{valid}} s_1 \neq T \ x\texttt{;}$$

B. Write big-step operational semantic rules or syntax-directed interpretation code (or pseudocode) for the statement constructs of Question 1. The environment must be made explicit. You can refer to the expression evaluation judgment or function without defining it. (5p)

SOLUTION:

$$\frac{\gamma \vdash e \Downarrow \langle \mathit{true}, \gamma' \rangle \qquad \gamma' \vdash s \Downarrow \gamma'' \qquad \gamma'' \vdash \texttt{while (}e\texttt{) sl} \Downarrow \gamma'''}{\gamma \vdash \texttt{while (}e\texttt{) } s\texttt{;} \Downarrow \gamma'''}$$

$$\frac{\gamma \vdash e \Downarrow \langle \mathit{false}, \gamma' \rangle}{\gamma \vdash \texttt{while (}e\texttt{) } s\texttt{;} \Downarrow \gamma'}$$

$$\frac{\gamma \vdash e \Downarrow \langle v, \gamma' \rangle}{\gamma \vdash e\texttt{;} \Downarrow \gamma'}$$

$$\frac{}{\gamma \vdash T \ x\texttt{;} \Downarrow \gamma, x := \mathit{null}}$$

Question 4 (Parsing): Show a BNF grammar for expressions with the constructs addition, multiplication, identifiers and parentheses. Associativity and precedence should follow the C standard. Apart from the built-in `Ident` token type, BNFC short-hands such as `coercions` must not be used. (4p)

Trace the LR-parsing of the expression $x + y * z + w$. Show how the stack and the input evolves and which actions are performed. (6p)

SOLUTION:

```
Exp2 ::= Ident
Exp1 ::= Exp1 "*" Exp2
Exp  ::= Exp "+" Exp1

Exp  ::= Exp1
Exp1 ::= Exp2
Exp2 ::= "(" Exp ")"

stack                input           actions
                     x + y * z + w   s
x                    + y * z + w     r
Exp2                 + y * z + w     r
Exp1                 + y * z + w     r
Exp                  + y * z + w     s
Exp +                y * z + w       s
Exp + y              * z + w         r
Exp + Exp2           * z + w         r
Exp + Exp1           * z + w         s
Exp + Exp1 *         z + w           s
Exp + Exp1 * z       + w             r
Exp + Exp1 * Exp2    + w             r
Exp + Exp1           + w             r
Exp                  + w             s
Exp +                w               s
Exp + w                              r
Exp + Exp2                           r
Exp + Exp1                           r
Exp                                  accept
```

Question 5 (Compilation):

A. Compile the following program into JVM-like assembler:

```
int x; bool b; x = 0; while (++x < 5) b = x < 3;
```

It is not necessary to remember exactly the names of the JVM instructions – only what arguments they take and how they work. (4p)

SOLUTION:

```
                        // int x; bool b;
        ldc 0           // x = 0;
        dup
        istore 0
        pop
TEST:   bipush 1
        iload 0         // while (++x ..
        bipush 1
        iadd
        dup
        istore 0
        ldc 5           // < 5) ..
        if_icmplt TRUE
        pop
        bipush 0
TRUE:   ifeq END
        bipush 1        // b = x < 3;
        iload 0
        ldc 3
        if_icmplt TRUE2
        pop
        bipush 0
TRUE2:  istore 1
        goto TEST
END:
```

B. Give the small-step semantics of the JVM instructions necessary to compile the program in the first part of this question. (6p)

SOLUTION: For each command, we give a transition $(P, V, S) \rightarrow (P', V', S')$ from old program counter P to its new value P', old variable store V to new store V', and old stack state S to new stack state

S'. Stack $S.v$ shall mean that the top value on the stack is v, the rest is S. Jump targets L are used as instruction addresses, and $P+1$ is the instruction address following P.

instruction	state before		state after	
`goto` L	(P, V, S)	$\to$	(L, V, S)	
`if_icmplt` L	$(P, V, S.v.w)$	$\to$	(L, V, S)	if $v < w$
`if_icmplt` L	$(P, V, S.v.w)$	$\to$	$(P+1, V, S)$	if $v \geq w$
`if_eq` L	$(P, V, S.v.w)$	$\to$	(L, V, S)	if $v = w$
`if_eq` L	$(P, V, S.v.w)$	$\to$	$(P+1, V, S)$	if $v \neq w$
`iload` a	(P, V, S)	$\to$	$(P+1, V, S.V(a))$	
`istore` a	$(P, V, S.v)$	$\to$	$(P+1, V[a := v], S)$	
`ldc` i	(P, V, S)	$\to$	$(P+1, V, S.i)$	
`bipush` i	(P, V, S)	$\to$	$(P+1, V, S.i)$	
`iadd`	$(P, V, S.v.w)$	$\to$	$(P+1, V, S.(v+w))$	
`dup`	$(P, V, S.v)$	$\to$	$(P+1, V, S.v.v)$	
`pop`	$(P, V, S.v)$	$\to$	$(P+1, V, S)$	

Question 6 (Functional languages): Show the big-step operational semantics inference rules (not as code) for a functional language with the expression constructs application, λ-abstraction, variables, integer literals and integer addition. The evaluation strategy should be call-by-name. Use closures and explicit environment. (6p)

Show the derivation tree (using your operational semantics) of the evaluation of the expression

```
(\x -> \y -> x + y) 3 4
```

(4p)

SOLUTION:

$$\frac{\gamma \vdash f \Downarrow (\lambda x.e)\{\delta\} \qquad \delta, x := a\{\gamma\} \vdash e \Downarrow v}{\gamma \vdash f\,a \Downarrow v}$$

$$\frac{}{\gamma \vdash \lambda x.e \Downarrow (\lambda x.e)\{\gamma\}}$$

$$\frac{}{\Gamma \vdash x \Downarrow v} x := v \in \gamma$$

$$\frac{}{\gamma \vdash i \Downarrow i}$$

$$\frac{\gamma \vdash e_1 \Downarrow i_1 \qquad \gamma \vdash e_2 \Downarrow i_2}{\gamma \vdash e_1 + e_2 \Downarrow i_1 + i_2}$$

$$\dfrac{\dfrac{\dfrac{}{\vdash \lambda x.\lambda y.x + y \Downarrow (\lambda x.\lambda y.x + y)\{\}} \qquad \dfrac{}{x := 3\{\} \vdash \lambda y.x + y \Downarrow (\lambda y.x + y)\{x := 3\{\}\}}}{\vdash (\lambda x.\lambda y.x + y)\,3 \Downarrow (\lambda y.x + y)\{x := 3\{\}\}} \qquad \dfrac{\dfrac{}{x := 3\{\}, y := 4\{\} \vdash x \Downarrow 3} \qquad \dfrac{}{x := 3\{\}, y := 4\{\} \vdash y \Downarrow 4}}{x := 3\{\}, y := 4\{\} \vdash x + y \Downarrow 7}}{\vdash (\lambda x.\lambda y.x + y)\,3\,4 \Downarrow 7}$$