Assignment 1

Author: Matthias Tilsner
Student-ID: 200882645
Course: COMP 6711

Submission: January 27, 2009

1 Design an attribute grammar to translate conditional arithmetic expressions

1.1 Expressions

$E_0 \rightarrow E_1$ + T

E_1.temp	= E_0.temp
E_1.loc	= E_0.loc
T.temp	= E_0.temp + 1
T.loc	= E_0.loc + E_1.length
E_0.code	= E_1.code [STORE E_0.temp] T.code [ADD E_0.temp]
E_0.length	= E_1.length + T.length + 2

E → T

T.temp	= E.temp
T.loc	= E.loc
E.code	= T.code
E.length	= T.length

1.2 Terms

$T_0 \rightarrow T_1$ * F

T_1.temp	= T_0.temp
T_1.loc	= T_0.loc
F.temp	= T_0.temp + 1
F.loc	= T_0.loc + T_1.length
T_0.code	= T_1.code [STORE T_0.temp] F.code [MULT T_0.temp]
T_0.length	= T_1.length + F.length + 2

T → F

F.temp	= T.temp
F.loc	= T.loc
T.code	= F.code
T.length	= F.length

1.3 Factors

F → i

F.code	=	[LOAD #i.name]
F.length	=	1

F → (E)

E.temp	=	F.temp
E.loc	=	F.loc
F.code	=	E.code
F.length	=	E.length

F → if L then E_0 else E_1

L.temp	=	F.temp
L.loc	=	F.loc
L.true	=	F.loc + L.length
L.false	=	F.loc + L.length + 1 + E_0.length
E_0.temp	=	F.temp + 1
E_0.loc	=	F.loc + L.length
E_1.temp	=	F.temp + 2
E_1.loc	=	F.loc + L.length + 1 + E_0.length
F.code	=	L.code E_0.code [JUMP *(F.loc + L.length + E_0.length + 1 + E_1.length)*] E_1.code
F.length	=	L.length + E_0.length + 1 + E_1.length

2 Design an attribute grammar to translate case statements

For this exercise I assume that the symbol * in JUMP instruction refers to the location of the current location, meaning not the location of the JUMP instruction itself, but of the instruction following the JUMP statement. The command [JUMP *+1] consequently means *jump to the instruction after next, thus skip the following instruction* Furthermore, I introduce the following mnemonics:

- JUMPNE: JUMP on not equal to comparator

2.1 Case Statements

Cs → case E of Cl

E.temp	=	Cs.temp + 1
E.loc	=	Cs.loc
Cl.temp	=	Cs.temp + 2
Cl.loc	=	Cs.loc + E.length + 1
Cl.skipOnSuccess	=	0
Cl.comp	=	Cs.temp
Cs.code	=	E.code [STORE Cs.temp] Cl.code
Cs.length	=	E.length + 1 + Cl.length

2.2 Case lists

$Cl_0 \rightarrow Cl_1$; I

Cl_1.temp	=	Cl_0.temp
Cl_1.loc	=	CL_0.loc
Cl_1.comp	=	Cl_0.comp
Cl_1.skipOnSuccess	=	I.length
I.temp	=	Cl_0.temp + 1
I.loc	=	Cl_0.loc + Cl_1length + 1
I.skipOnSuccess	=	Cl_0.skipOnSuccess
I.comp	=	Cl_0.comp
Cl_0.code	=	Cl_1.code I.code
Cl_0.length	=	Cl_1.length

Cl → I

I.temp	=	Cl.temp
I.loc	=	Cl.loc
I.skipOnSuccess	=	Cl.skipOnSuccess
I.comp	=	Cl.comp
Cl.code	=	I.code
Cl.length	=	I.length

2.3 Case items

I → n : St

St.temp	=	I.temp
St.loc	=	I.loc + 3
I.code	=	[LOAD #n.name] [COMP I.comp] [JUMPNE *(*+St.length+1)*] St.code [JUMP *(*+I.skipOnSuccess)*]
I.length	=	4 + St.length

3 Define an attribute grammar to translate C-Style expressions

For the following exercise I introduce the following mnemonics:

- INC increment the value of the accumulator by one
- DEC decrement the value of the accumulator by one

3.1 Expressions

$E_0 \rightarrow E_1 + E_2$

E_1.temp	=	E_0.temp
E_1.loc	=	E_0.loc
E_2.temp	=	E_0.temp + 1
E_2.loc	=	E_0.loc + E_1.length
E_0.code	=	E_1.code [STORE E_0.temp] E_2.code [ADD E_0.temp]
E_0.length	=	E_1.length + 2 + E_2.length

$E_0 \rightarrow E_1$ * E_2

E_1.temp	=	E_0.temp
E_1.loc	=	E_0.loc
E_2.temp	=	E_0.temp + 1
E_2.loc	=	E_0.loc + E_1.length
E_0.code	=	E_1.code [STORE E_0.temp]
		E_2.code [MULT E_0.temp]
E_0.length	=	E_1.length + 2 + E_2.length

$E_0 \rightarrow$ (E_1)

E_1.temp	=	E_0.temp
E_1.loc	=	E_0.loc
E_0.code	=	E_1.code
E_0.length	=	E_1.length

$E_0 \rightarrow$ i

E_0.code	=	[LOAD #i.name]
E_0.length	=	1

$E_0 \rightarrow$ i := E_1

E_1.temp	=	E_0.temp
E_1.loc	=	E_0.loc
E_0.code	=	E_1.code [STORE #i.name]
E_0.length	=	E_1.length + 1

$E_0 \rightarrow$ i ++

E_0.code	=	[LOAD #i.name] [INC] [STORE #i.name] [DEC]
E_0.length	=	4

$E_0 \rightarrow$ ++ i

E_0.code	=	[LOAD #i.name] [INC] [STORE #i.name]
E_0.length	=	3

4 Design an attribute grammar to translate loop statements

For this exercise I introduce the following mnemonics:

- JUMPG: JUMP on greater than comparator

4.1 Loop statements

Lp $\rightarrow$ for i=E_0 to E_1 step E_2 do St

E_0.temp	=	Lp.temp + 1
E_0.loc	=	Lp.loc + E_1.length + 1
E_1.temp	=	Lp.temp + 2
E_1.loc	=	Lp.loc
E_2.temp	=	Lp.temp + 3
E_2.loc	=	Lp.loc + E_0.length + E_1.length + St.length + 4
St.temp	=	Lp.temp + 4
St.loc	=	Lp.loc + E_0.length + E_1.length + 4
Lp.code	=	E_1.code [STORE Lp.temp] E_0.code [STORE #i.name] [COMP Lp.temp] [JUMPG *(Lp.loc + Lp.length)*] St.code E_2.code [ADD #i.name] [JUMP *(Lp.loc + 3)*]
Lp.length	=	E_0.length + E_1.length + E_2.length + St.length + 6

5 Design an attribute grammar to translate gosub expressions

For this exercise I introduce the following mnemonics:

- LOADVAL: LOAD provided parameter into accumulator
- IJUMP: JUMP to value stored at address provided as parameter
- JUMPZ: JUMP when accumulator is equal to zero

There are two different options for translating the goto statement. Option 1 assumes that labels may be used for jumps. Option 2 assumes, that all labeled statements are defined before using them and that *i* is has a variable with an assigned storage available at #i.loc assigned by the token parser. All occurences of the identical *i* must have the same #i.loc value, thus referencing the identical variable storage. In this option, the labeled statement will not be executed upon declaration, but only when it is called.

5.1 Expressions - Option 1

E → gosub i

E.code	=	[LOADVAL *(E.loc+3)*] [PUSH] [JUMP #i.name]
E.length	=	3

5.2 Expressions - Option 2

E → gosub i

E.code	=	[LOADVAL *(E.loc+3)*] [PUSH] [IJUMP #i.loc]
E.length	=	3

5.3 Labeled statements - Option 1

Lst → i St

St.temp	=	Lst.temp
St.loc	=	Lst.loc
Lst.code	=	[#i.name NOOP] St.code
Lst.length	=	St.length + 1

5.4 Labeled statements - Option 2

Lst → i St

St.temp	=	Lst.temp
St.loc	=	Lst.loc
Lst.code	=	[LOADVAL *(Lst.loc+2)*] [STORE #i.loc] [JUMP *(Lst.loc + Lst.temp)*] St.code
Lst.length	=	St.length + 3

5.5 Return statements

Rs → return (E)

E.temp	=	Rs.temp + 1
E.loc	=	Rs.loc + 2
Rs.code	=	[POP] [JUMPZ *(*+ E.length + 2)*] [STORE Rs.temp] E.code [JUMP Rs.temp] [PUSH]
Rs.length	=	E.length + 5