Programming Languages: Principles and Paradigms

Allen Tucker and Robert Noonan

Errata list for first printing (October 2001) - lists all corrections as of January 20, 2003. (Negative line numbers mean "from bottom of page.")

Page	Line	Change from:	to:
5	2	[Linda 1980]	[Carriero 1980]
	3	[HPF 1995]	[Adams 1997]
	-8	[K&R 1988]	[Kernighan 1988]
19	-5	[Naur 1960]	[Naur 1963]
27	-2	`[a-za-Z ]`	`[a-zA-Z0-9!-~ ]`
29	7	an *in-order*	a *post-order*
	10	an in-order	a post-order
31	10	an inorder	a post-order
35	-13	about 8	about 7
	-12	18	16
40	Fig 2.17	`Term term () {...}`	`Expression term () {...}`
		`Factor factor () {...}`	`Expression factor () {...}`
43	-3	begins with 1, 1,	begins with 0, 1, 1,
	-2	fib(n) = 1 if n = 0 or 1	fib(n) = n if n = 0 or 1
46	Ex 2.3	tokenStream	TokenStream
51	-21	raise	cause
	-19	raise	cause
62	-11	Q[s.target\s.source]	*Q*[s.source\s.target]
63	5	$(m = max(a, b)\}$	$\{m = max(a, b)\}$
66	14	$1 \le i \wedge i \le n$	$1 \le i + 1 \wedge i + 1 \le n$
81	6	$M: Division \times \Sigma \rightarrow \Sigma$	$M: Division \times \Sigma \rightarrow Value$
	12	these expression	these expressions
	13	`M(z+2)*y,`	`M((z+2)*y,`

Page	Line	Change from:	to:
91	-1	`1 1000 0000 101 1000 0000 0000 0000`	`0 1000 0000 101 1000 0000 0000 0000`
94	2	n/a (first occurrence)	`.`
Tbl 4.3			
	3	n/a (first occurrence)	`sizeof`
	5	n/a n/a n/a n/a	n/a n/a `new`
	6	`() sizeof`	`()`
		`()` n/a	n/a `()`
	7	`()` n/a	`()`
	8	, (2 times)	n/a (2 times)
100	-5	program.	program. (Technically, `%` is not a Jay operator, but for the purpose of this example we will assume it is.)
105	13	`i+1`	`i+increment`
107	6	*CaseHead Cases*	*CaseHead Case*
	8	*Statements*	*Statement*
	12, 13	*CaseHead* followed by a list of statements	series of *CaseHeads* followed by a statement
114	-2	global variables... and their types	global variables and methods– that is, all pairs of globals and their types, and all pairs of methods and return types.
115	1	$tm_g = \{\langle h, int\rangle, \langle i, int\rangle\}$	$tm_g = \left\{\begin{array}{c}\langle h, int\rangle, \langle i, int\rangle, \langle A, \text{void}\rangle, \\ \langle B, \text{void}\rangle, \langle main, \text{void}\rangle\end{array}\right\}$
	13	declarations *g*	declarations and methods *g*,
	23	$V(\text{p.globals})$	$V(tm_g)$
	24	$typing_m(\text{p.globals},$	$typing_m(tm_g,$
	25	global variable	global variable and method

Page	Line	Change from:	to:
	29	$\{\langle h, int\rangle, \langle i, int\rangle\}$	tm_g
	30	$\{\langle h, int\rangle, \langle i, int\rangle\}$	tm_g
	31	$\{\langle h, int\rangle, \langle i, int\rangle\}$	tm_g
	32	$\{\langle h, int\rangle, \langle i, int\rangle\}$	tm_g
121	23	the product	a composite
	25	given by $\gamma_m(v)$, and the	defined by the function $\gamma_m(v) = max\ x\colon \langle v, x\rangle \in \gamma_m$. The
122	1-3	a value in ... method *m*.	the largest *x* in the range {0, ..., *a* - 1} for variable *v* in which $\langle v, x\rangle \in \gamma_m$.
125	19	$\gamma_g\ \overline{\cup}$	$\gamma_g \cup$
	31	$\gamma_g\ \overline{\cup}$	$\gamma_g \cup$
128	-2	$\sigma - c.\text{params} - c.\text{locals}$	σ
129	1-6	removes... That is:	effectively makes *m*'s parameters and locals most visible by assigning them the largest (maximum) addresses. Deactivation of a call reverses the effect of the activation by removing the stack frame thus created and making the calling method's parameters and locals most visible again. That is:
	7	$\sigma)\ \overline{\cup}\ c.\text{params}\ \overline{\cup}\ c.\text{locals}$	$\sigma)$
	17	$\gamma_{main} - \{\langle a, 2\rangle, \langle b, 3\rangle\}\ \overline{\cup}$	$\gamma_{main} \cup$
	18	$\langle h, 0\rangle,$	$\langle h, 0\rangle, \langle i, 1\rangle, \langle a, 2\rangle, \langle b, 3\rangle,$
	23	is declared locally within A	has a higher address within γ_A

Page	Line	Change from:	to:
	24-25	are also removed ... call.	are also inaccessible to A, provided that strong type checking rules are employed.
	29	$\gamma_A - \{\langle x, 4\rangle, \ldots, \langle j, 7\rangle\} \ \overline{\cup}$	$\gamma_A \cup$
	30	$\langle i, 1\rangle,$	$\langle i, 1\rangle, \langle a, 2\rangle, \langle b, 3\rangle, \langle x, 4\rangle,$ $\langle y, 5\rangle, \langle i, 6\rangle, \langle j, 7\rangle,$
130	2	$\langle h, 0\rangle,$	$\langle h, 0\rangle, \langle i, 1\rangle, \langle a, 2\rangle, \langle b, 3\rangle,$
133	13	/	$\wedge$ (two times)
	16	`[-5]`	`[-5];`
139	-3	$\langle a, undef\rangle$	$\langle b, undef\rangle$
		$\langle a + 1, undef\rangle$	$\langle b + 1, undef\rangle$
		$\langle a + k - 1, undef\rangle$	$\langle b + k - 1, undef\rangle$
140	-3	$\mu \ \overline{\cup} \ \{\langle a + i, new(d_i.\text{size})\rangle\}$	$\mu_1 \ \overline{\cup} \ \{\langle a + i, b\rangle\}$ where $\langle b, \mu_1\rangle = new(d_i.\text{size}, \mu)$
141	Fig 5.11	@A[0]	γ(A[0])
	2	`int[10])`	`int[10];`
	10	$delete(a + i, d_i.\ size)$	$delete(a + i, d_i.\ size, \mu)$ $\overline{\cup} \ \{\langle a + i, unused\rangle\}$
	11	$\mu \ \overline{\cup} \ \{\langle a + i, unused\rangle\}$	$\mu \ \overline{\cup} \ \{\langle a + i, unused\rangle\}$ otherwise
142	Fig 5.12	@p.x	γ(p.x)
		@p.y	γ(p.y)
	-11	$b = @new(k)$	$(b, \mu') = new(k, \mu)$
145	12	`p->next`	`p.next`
151	Ex 5.1a	μ	σ
	Ex 5.1b	$\mu(\mu(x))$	$\mu(\gamma(x))$
	Ex 5.1f	μ	σ
152	15	`search`	`binsearch`

Page	Line	Change from:	to:
	17	`search`	`binsearch`
	20	`int result;`	`boolean result;`
	-7	`[[2..7];`	`[2..7];`
153	25	$\mu\ \overline{\mathrm{U}}\ \{\langle a+i, unused\rangle\}$	$\mu\ \overline{\mathrm{U}}\ \{\langle a+i, unused\rangle\}$ otherwise
172	23-24	`int pop(STACK` `void push(STACK`	`int pop(STACK*` `void push(STACK*`
173	9	`int pop(STACK`	`int pop(STACK*`

	12-15	`if (!empty()) {`
		`  rslt=stack->val;`
		`  tmp=stack;`
		`  stack=stack->val;`
			`if (!empty(*stack)) {`
			`  rslt=(*stack)->val;`
			`  tmp=*stack;`
			`  *stack=(*stack)->val;`

	20	`void push(STACK stack,`	`void push(STACK* stack,`

	21	`STACK tmp = (STACK)`
		`  malloc(sizeof(struct`
		`  Node));`
			`STACK tmp;`
			`tmp=(STACK)malloc(sizeof(`
			`  struct Node));`

	23-24	`tmp->next=stack;` `stack=tmp;`	`tmp->next=*stack;` `*stack=tmp;`
	27	`if (!empty()) {`	`if (!empty(*stack)) {`
187	Fig 7.15	`public intVal`	`public int intVal`
	Fig 7.16	`public isUndef( )` `  { returns true; }`	`public boolean isUndef( )` `  { return true; }`
200	3	[2000]	[2001]
201	3	Meyers	Meyer
202	9	`!empty(stack)`	`!empty( )`
208	-10	$[\lambda x$	$(\lambda x$
209	4	(abz)	abz
	7	$(\lambda y \ldots b$	$((\lambda y \ldots b)$
		$(\lambda x \ldots a$	$((\lambda x \ldots a)$
217	-7	`(length ())`	`(length '())`
218	24	`(if (null? alist) ()`	`(if (null? alist) '()`
219	3	`(if (null? alist) ()`	`(if (null? alist) '()`

Page	Line	Change from:	to:
	4-5	`(if (equal? x) ...` `  (cons y ...`	`(if (equal? y) ...` `  (cons x ...`
	20	`(if (null? alist) ()`	`(if (null? alist) ‘()`
	22-23	`(if (equal? x) ...` `  (cons y ...`	`(if (equal? y) ...` `  (cons x ...`
	-3	`(if (null? alist) ()`	`(if (null? alist) ‘()`
222	21	`equal`	`equal?`
	22	`equal`	`equal?`
	-3	`equal`	`equal?`
	-2	`equal`	`equal?`
225	9	`gett`	`get`
226	18	`(atom? expr)`	`(not (list? expr))`
232	26	`(if (null? lst) ()`	`(if (null? lst) ‘()`
233	4	`(if (null? lst) ()`	`(if (null? lst) ‘()`
233	24	[Haskell 1999]	[Thompson 1999]
245	3	`Block`	`Block BlockType`
	9	`Block` (1st occurrence)	`Blocktype`
		`Block` (2nd occurrence)	`Blocktype deriving (Show)`
248	Ex 8.1c	$(\lambda v \cdot (\lambda w \cdot w)((\lambda x \cdot x)y(\lambda z \cdot z)))$	$((\lambda v \cdot (\lambda w \cdot w))((\lambda x \cdot x)(y(\lambda z \cdot z))))$
249	Ex 8.12	nx^{n-1}	nu^{n-1}
250	Ex 8.25	unary operator `not`	Jay unary operator `"!"`
	Ex 8.26	`eval("+" y 2)...`	`eval(Binary "+" (Var y)` `  (Lit (Intval 2)))...`
251	Ex 8.30	Rewrite the... Haskell lists.	Give a complete... Haskell recursive data types.
	Ex 8.31	Rewrite the... Haskell lists.	Give a complete... Haskell recursive data types.
257	19	only if	if
283	-11	`fib(0, 1).`	`fib(0, 0).`

Page	Line	Change from:	to:
	-8	8th	9th
	-6	Answer = 8	Answer = 9
	-4	`? - fib(8, Answer).`	`? - fib(9, Answer).`
284	5	`fib(0, 1).`	`fib(0, 0).`
	8	Answer = 8	Answer = 9
285	Ex 9.10	nx^{n-1}	nu^{n-1}
287	-4	... in Exercise 9.20	... in Exercise 9.17
288	27	`[c,b,d,a]`	`[c,b,d,a], Answer`
308	26	`lastX, lastY`	`Math.min(lastX, x),` `Math.min(lastY, y)`
330	Fig 11.3	`in`	`inn`
	4, 16, 17		
331	-14	`V(nonempty);`	`signal(nonempty);`
344	-2	Sleepy	Sleeping
348	12	<*w*, 9>	<*w*, 4>
361	12	}	{
375	19-21	However, if ... disappears.	(Delete this sentence.)
380	-8	However, in	However, if
	-5	Using the ... program.	(Delete this sentence.)
387	2	`choice = new Button();`	`choice = new Choice();`

Additional Bibliography Entries

1. Carriero, Nicholas and David Gelernter. [1990]. *How To Write Parallel Programs: A First Course.* MIT Press.
2. Cooper, James W. [2000]. *Java Design Patterns: A Tutorial*. Addison-Wesley.
3. Hoare, C. A. R. [1974]. Monitors: an operating system concept. *Communications of ACM*, 17 (October 1974), pp. 549-557.
4. Horstmann, Cay S. [1995]. *Mastering Object-Oriented Design in C++*. John Wiley.

5. Liskov, B. H. and J. Guttag [2001]. *Program Development in Java*. Addison-Wesley.
6. Perry, Jo Ellen and Harold D. Levin [1996]. *An Introduction to Object-Oriented Design in C++*. Addison-Wesley.
7. Pountain, Dick [1995]. Constraint logic programming. *Byte*, 20, 2, pp. 159-160.
8. Thompson, Simon [1999]. *Haskell: The Craft of Functional Programming.* Addison-Wesley.