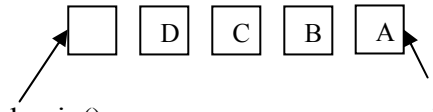
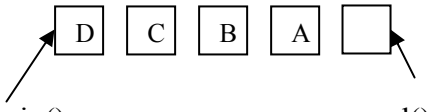
И. А. Волкова, А. В. Иванов, Л. Е. Карпов
Основы объектно-ориентированного программирования.
Язык программирования C++.

Список найденных опечаток

21.03.2011

№	Страница	Старое	Исправленное
1	стр.12, Пример:	int & imax (int **&** m){	int & imax (int ***** m){
2	стр.43, Пример: внизу	friend vector operator+= (**const vector** & v1, const vector & v2);	friend vector & operator+= (**vector** & v1, const vector & v2);
3	стр.44, сверху	vector operator+=(**const** vector & v1, const vector & v2);	vector **&** operator+= (vector & v1, const vector & v2);
4	стр. 50	float −> double **−> long double**	float −> double
5	стр. 54, f(a):	int −> long : шаг **2. Расширение (целочисленное)**	int −> long : **шаг 3. Стандартное преобразование**
6	стр. 64, 10.4. — третий абзац	При **открытом** наследовании ....., как и его собственные **открытые** члены	При **закрытом** наследовании ....., как и его собственные **закрытые** члены
7	стр. 67, конец примера сверху	// на объект **c1** типа класс **C**	// на объект **a1** типа класс **A**, **// в общем случае такой вызов некорректен**
8	стр. 72, первый абзац	... на который **этот указывает** указатель	... на который **указывает этот** указатель
9	стр. 73, раздел 11.1, первый абзац, 3-е предложение	**сначала** срабатывает деструктор базового класса ..... , а **затем** срабатывает деструктор текущего	**во вторую очередь** срабатывает деструктор базового класса ..... , а **сначала** срабатывает деструктор текущего ....

10	стр. 99, class allocator	void dea**l**locate (pointer p, size_type n);
		// **перераспределение** n объектов типа T
		
		...
		void destroy (pointer p); // **освобождает** память, на которую указывает p
		...
		}
			void dea**ll**ocate (pointer p, size_type n);
			// **освобождает память для** n объектов типа T **без вызова деструкторов**
			...
			void destroy (pointer p); // **вызывает деструктор для *p, не освобождая** память, на которую указывает p
			};

11	стр. 100, рисунок для обратных итерато-ров		

12	стр. 106, iterator erase (...	iterator erase (iterator i) { ... return (i); }
		// уничтожение заданного элемента и выдача элемента,
		// следующего за удалённым
		
		iterator erase (iterator start, iterator finish)
		//уничтожение диапазона [start,finish) и выдача
		// следующего за последним удалённым
			iterator erase (iterator i) { ... return (res); }
			// уничтожение заданного элемента и выдача **итератора** элемента,
			// следующего за удалённым
			
			iterator erase (iterator start, iterator finish)
			// уничтожение диапазона [start,finish) и выдача
			// **итератора элемента,** следующего за последним удалённым

13	стр. 106, внизу	template <class C> typename C::**iterator** find_last(...{
		typename C::**iterator** p = c.end ();
		...
			template <class C> typename C::**const_iterator** find_last...{
			typename C::**const_iterator** p = c.end (); ...

14	стр. 107, сверху	template <class C> typename C::**iterator** find_last(... { typename C::**reverse_iterator** ri = find(...
		...
		
		typename C::**iterator** i = ri.base ();
			template <class C> typename C::**const_iterator** find_last(... {
			typename C::**const_reverse_iterator** ri = find(...
			...
			typename C::**const_iterator** i = ri.base ();