

Содержание

Предисловие	24
Введение	25
Новые концепции программирования	25
Объектно-ориентированное программирование	25
Унифицированный язык моделирования	26
Языки и платформы разработки	26
Для чего нужна эта книга	27
Новые концепции	27
Последовательность изложения материала	27
Знания, необходимые для чтения этой книги	28
Техническое и программное обеспечение	28
Консольные программы	28
Исходные тексты программ	28
Упражнения	29
Проще, чем кажется	29
Преподавателям	29
Стандартный C++	29
Унифицированный язык моделирования (UML)	29
Средства разработки программного обеспечения	30
Различия между C и C++	30
Оптимальный порядок изучения ООП	30
Нововведения в C++	31
Избыточные возможности	31
Упражнения	31
От издательства	31
Глава 1. Общие сведения	32
Для чего нужно объектно-ориентированное программирование?	32
Процедурные языки	32
Деление на функции	33
Недостатки структурного программирования	33
Неконтролируемый доступ к данным	34
Моделирование реального мира	35
Объектно-ориентированный подход	36
Аналогия	37
ООП: подход к организации программы	38

Характеристики объектно-ориентированных языков	38
Объекты	38
Классы	39
Наследование	40
Повторное использование кода	42
Пользовательские типы данных	42
Полиморфизм и перегрузка	42
С++ и С	43
Изучение основ	44
Универсальный язык моделирования (UML)	44
Резюме	45
Вопросы	46

Глава 2. Основы программирования на С++ 48

Что необходимо для работы	49
Структура программы	49
Функции	49
Операторы	51
Разделяющие знаки	51
Вывод с использованием cout	52
Строковые константы	53
Директивы	53
Директивы препроцессора	54
Заголовочные файлы	54
Директива using	55
Комментарии	55
Синтаксис комментариев	55
Использование комментариев	56
Альтернативный вид комментариев	56
Переменные целого типа	56
Описание переменных целого типа	57
Объявление и определение переменной	58
Имена переменных	59
Операция присваивания	59
Целые константы	59
Оператор вывода	60
Манипулятор endl	60
Другие целые типы	61
Символьные переменные	61
Символьные константы	62
Инициализация	63
Управляющие последовательности	63
Ввод с помощью cin	64
Определение переменных при первом использовании	65
Каскадирование операции <<	66
Выражения	66
Приоритеты выполнения операций	66
Вещественные типы	67
Тип float	67

Типы double и long double	68
Вещественные константы	68
Префикс const	69
Директива #define	70
Тип bool	70
Манипулятор setw	71
Каскадирование операции <<	72
Множественное определение	72
Файл заголовка IOMANIP	73
Таблица типов переменных	73
Беззнаковые типы данных	73
Преобразования типов	75
Неявные преобразования типов	75
Явные преобразования типов	77
Арифметические операции	78
Остаток от деления	79
Арифметические операции с присваиванием	79
Инкремент	81
Декремент	82
Библиотечные функции	82
Заголовочные файлы	83
Библиотечные файлы	84
Заголовочные и библиотечные файлы	84
Формы директивы #include	84
Резюме	85
Вопросы	87
Упражнения	88

Глава 3. Циклы и ветвления 92

Операции отношения	92
Циклы	94
Цикл for	94
Инициализирующее выражение	96
Условие выполнения цикла	96
Инкрементирующее выражение	96
Число выполнений цикла	97
Несколько операторов в теле цикла	97
Блоки и область видимости переменных	98
Форматирование и стиль оформления циклов	99
Обнаружение ошибок	100
Варианты цикла for	100
Определение счетчика цикла внутри оператора цикла for	101
Несколько инициализирующих выражений и условий цикла	101
Цикл while	102
Несколько операторов в цикле while	104
Приоритеты арифметических операций и операций отношения	105
Цикл do	106
Выбор типа цикла	108
Ветвления	108

Условный оператор if	109
Несколько операторов в теле if	110
if внутри циклов	110
Функция exit().	111
Оператор if...else	112
Функция getche()	113
Условия с присваиванием	114
Вложенные ветвления if...else	116
if и else во вложенных ветвлениях	117
Конструкция else...if	119
Оператор switch	119
Оператор break	121
switch и символьные переменные	122
Ключевое слово default	123
Сравнение switch и if...else	124
Условная операция	124
Логические операции	127
Операция логического И	127
Логическое ИЛИ	128
Логическое НЕ	129
Целые величины в качестве булевых	129
Приоритеты операций C++	130
Другие операторы перехода	131
Оператор break	131
Расширенная таблица символов ASCII	133
Оператор continue	133
Оператор goto	134
Резюме	135
Вопросы	136
Упражнения	138
Глава 4. Структуры	142
Структуры	142
Простая структура	143
Определение структуры	143
Определение структурной переменной	144
Доступ к полям структуры	146
Другие возможности структур	146
Пример применения структур	148
Вложенные структуры	150
Пример карточной игры	154
Структуры и классы	156
Перечисления	156
Дни недели	157
Перечисления и программа подсчета числа слов	159
Пример карточной игры	161
Недостаток перечислений	162
Примеры перечисляемых типов	163
Резюме	163
Вопросы	164
Упражнения	165

Глава 5. Функции.	168
Простые функции	169
Объявление функции	170
Вызов функции	171
Определение функции	171
Обычные и библиотечные функции	172
Отсутствие объявления	173
Передача аргументов в функцию	174
Передача констант в функцию	174
Передача значений переменных в функцию	175
Передача аргументов по значению	176
Структурные переменные в качестве аргументов	177
Имена переменных внутри прототипа функции	181
Значение, возвращаемое функцией	181
Оператор return	182
Исключение ненужных переменных	184
Структурная переменная в качестве возвращаемого значения	185
Ссылки на аргументы	186
Передача по ссылке аргументов стандартных типов	187
Усложненный вариант передачи по ссылке	189
Передача структурных переменных по ссылке	191
Замечание о ссылках	192
Перегруженные функции	192
Переменное число аргументов функции	193
Различные типы аргументов	195
Рекурсия	196
Встраиваемые функции	198
Аргументы по умолчанию	200
Область видимости и класс памяти	202
Локальные переменные	203
Глобальные переменные	205
Статические локальные переменные	207
Возвращение значения по ссылке	208
Вызов функции в качестве левого операнда операции присваивания	209
Зачем нужно возвращение по ссылке?	210
Константные аргументы функции	210
Резюме	212
Вопросы	213
Упражнения	215
Глава 6. Объекты и классы	217
Простой класс	217
Классы и объекты	218
Определение класса	219
Использование класса	222
Вызов методов класса	222
Объекты программы и объекты реального мира	224
Детали изделия в качестве объектов	224
Круги в качестве объектов	225

Класс как тип данных	226
Конструкторы	227
Пример со счетчиком	228
Графический пример	231
Объекты в качестве аргументов функций	232
Объекты в качестве аргументов	236
Конструктор копирования по умолчанию	237
Объекты, возвращаемые функцией	239
Аргументы и объекты	240
Пример карточной игры	242
Структуры и классы	244
Классы, объекты и память	245
Статические данные класса	247
Применение статических полей класса	247
Пример использования статических полей класса	247
Раздельное объявление и определение полей класса	248
const и классы	249
Константные методы	250
Константные объекты	252
Зачем нужны классы?	253
Резюме	254
Вопросы	255
Упражнения	257

Глава 7. Массивы и строки 261

Основы массивов	262
Определение массивов	263
Элементы массива	263
Доступ к элементам массива	263
Среднее арифметическое элементов массива	264
Инициализация массива	265
Многомерные массивы	267
Передача массивов в функции	271
Массивы структур	273
Массивы как члены классов	275
Массивы объектов	278
Массивы интервалов	278
Границы массива	280
Доступ к объектам в массиве	280
Массивы карт	281
Строки	284
Строковые переменные	285
Считывание нескольких строк	288
Копирование строк	289
Копирование строк более простым способом	290
Массивы строк	291
Строки как члены классов	292
Определенные пользователем типы строк	294
Стандартный класс string языка C++	296

Определение объектов класса <code>string</code> и присваивание им значений	296
Ввод/вывод для объектов класса <code>string</code>	298
Поиск объектов класса <code>string</code>	299
Модификация объектов класса <code>string</code>	300
Сравнение объектов класса <code>string</code>	301
Доступ к символам в объектах класса <code>string</code>	302
Другие методы класса <code>string</code>	303
Резюме	304
Вопросы	304
Упражнения	307
Глава 8. Перегрузка операций	312
Перегрузка унарных операций	313
Ключевое слово <code>operator</code>	314
Аргументы операции	315
Значения, возвращаемые операцией	315
Временные безымянные объекты	317
Постфиксные операции	318
Перегрузка бинарных операций	320
Арифметические операции	320
Объединение строк	323
Множественная перегрузка	325
Операции сравнения	325
Операции арифметического присваивания	328
Операция индексации массива (<code>[]</code>)	330
Преобразование типов	334
Преобразования основных типов в основные типы	335
Преобразования объектов в основные типы и наоборот	336
Преобразования строк в объекты класса <code>string</code> и наоборот	338
Преобразования объектов классов в объекты других классов	340
Преобразования: когда что использовать	346
Диаграммы классов UML	346
Объединения	347
Направленность	347
«Подводные камни» перегрузки операций и преобразования типов	348
Использование похожих значений	348
Использование похожего синтаксиса	348
Показывайте ограничение	349
Избегайте неопределенности	349
Не все операции могут быть перегружены	349
Ключевые слова <code>explicit</code> и <code>mutable</code>	349
Предотвращение преобразования типов с помощью <code>explicit</code>	350
Изменение данных объекта, объявленных как <code>const</code> , используя ключевое слово <code>mutable</code>	351
Резюме	353
Вопросы	353
Упражнения	356

Глава 9. Наследование 361

Базовый и производный классы	362
Определение производного класса	364
Обобщение в диаграммах классов в UML	364
Доступ к базовому классу	365
Результат программы COUNTEN	366
Спецификатор доступа protected	366
Недостатки использования спецификатора protected	368
Неизменность базового класса	368
Разнообразие терминов	368
Конструкторы производного класса	368
Перегрузка функций	370
Какой из методов использовать?	372
Операция разрешения и перегрузка функций	372
Наследование в классе Distance	373
Применение программы ENGLN	374
Конструкторы класса DistSign	375
Методы класса DistSign	375
В поддержку наследования	375
Иерархия классов	376
Абстрактный базовый класс	379
Конструкторы и функции	380
Наследование и графика	380
Общее и частное наследование	383
Комбинации доступа	383
Выбор спецификатора доступа	384
Уровни наследования	385
Множественное наследование	388
Методы классов и множественное наследование	389
Частное наследование в программе EMPMULT	393
Конструкторы при множественном наследовании	393
Конструкторы без аргументов	396
Конструктор со многими аргументами	396
Неопределенность при множественном наследовании	397
Включение: классы в классах	398
Включение в программе EMPCONT	399
Композиция: сложное включение	403
Роль наследования при разработке программ	403
Резюме	404
Вопросы	405
Упражнения	407

Глава 10. Указатели 411

Адреса и указатели	412
Операция получения адреса &	412
Переменные указатели	414
Недостатки синтаксиса	416
Указатели должны иметь значение	416

Доступ к переменной по указателю	417
Указатель на void	420
Указатели и массивы	421
Указатели-константы и указатели-переменные	423
Указатели и функции	424
Передача простой переменной.	424
Передача массивов	426
Сортировка элементов массива	428
Расстановка с использованием указателей	428
Сортировка методом пузырька	430
Указатели на строки	432
Указатели на строковые константы.	432
Строки как аргументы функций.	433
Копирование строк с использованием указателей	434
Библиотека строковых функций	434
Модификатор const и указатели	435
Массивы указателей на строки	436
Управление памятью: операции new и delete	437
Операция new	438
Операция delete	439
Класс String с использованием операции new	440
Указатели на объекты	442
Ссылки на члены класса	443
Другое применение операции new	444
Массив указателей на объекты.	445
Действия программы	446
Доступ к методам класса.	446
Связный список	447
Цепочка указателей	447
Добавление новых элементов в список	449
Получение содержимого списка	450
Классы, содержащие сами себя	450
Пополнение примера LINKLIST	451
Указатели на указатели	451
Сортируем указатели.	453
Тип данных person**	454
Сравнение строк	454
Пример разбора строки	455
Разбор арифметических выражений	456
Программа PARSE	457
Симулятор: лошадиные скачки	459
Разработка лошадиных скачек	460
Моделирование хода времени	463
Уничтожение массива указателей на объекты	463
Функция patch()	464
Диаграммы UML	464
Диаграмма состояний в UML	465
Состояния	466
Переходы	466
От состояния к состоянию	466

Отладка указателей	467
Резюме	467
Вопросы	469
Упражнения	471

Глава 11. Виртуальные функции 476

Виртуальные функции	476
Доступ к обычным методам через указатели.	477
Доступ к виртуальным методам через указатели.	479
Позднее связывание	481
Абстрактные классы и чистые виртуальные функции	481
Виртуальные функции и класс person	483
Виртуальные функции в графическом примере	485
Виртуальные деструкторы	488
Виртуальные базовые классы	489
Дружественные функции	491
Дружественные функции как мосты между классами	491
Ломающая стены	492
Пример с английскими мерами длины	493
Дружественность и функциональная запись	496
Дружественные классы	499
Статические функции	500
Доступ к статическим функциям	501
Инициализация копирования и присваивания	502
Перегрузка оператора присваивания	503
Конструктор копирования	506
Объектные диаграммы UML	510
Эффективное использование памяти классом String.	510
Указатель this	516
Доступ к компонентным данным через указатель this	517
Использование this для возврата значений	518
Исправленная программа STRMEM	520
Динамическая информация о типах	523
Проверка типа класса с помощью dynamic_cast	523
Изменение типов указателей с помощью dynamic_cast	524
Оператор typeid	526
Резюме	527
Вопросы	528
Упражнения	531

Глава 12. Потоки и файлы 536

Потоковые классы	536
Преимущества потоков.	537
Иерархия потоковых классов.	537
Класс ios	539
Класс istream	542
Класс ostream	543
Классы iostream и _withassign.	544
Предопределенные потоковые объекты	544

Ошибки потоков	545
Биты статуса ошибки	545
Ввод чисел	546
Переизбыток символов	547
Ввод при отсутствии данных	547
Ввод строк и символов	548
Отладка примера с английскими расстояниями	548
Потоковый ввод/вывод дисковых файлов	551
Форматированный файловый ввод/вывод	551
Строки с пробелами	554
Ввод/вывод символов	555
Двоичный ввод/вывод	557
Оператор <code>reinterpret_cast</code>	558
Закрытие файлов	558
Объектный ввод/вывод	559
Совместимость структур данных	560
Ввод/вывод множества объектов	561
Биты режимов	563
Указатели файлов	564
Вычисление позиции	564
Вычисление сдвига	565
Функция <code>tellg()</code>	567
Обработка ошибок файлового ввода/вывода	567
Реагирование на ошибки	567
Анализ ошибок	568
Файловый ввод/вывод с помощью методов	570
Как объекты записывают и читают сами себя	570
Как классы записывают и читают сами себя	572
Код типа объекта	578
Перегрузка операторов извлечения и вставки	581
Перегрузка <code>cout</code> и <code>cin</code>	581
Перегрузка <code><<</code> и <code>>></code> для файлов	583
Память как поток	585
Аргументы командной строки	586
Вывод на печатающее устройство	589
Резюме	590
Вопросы	591
Упражнения	592

Глава 13. Многофайловые программы 596

Причины использования многофайловых программ	596
Библиотеки классов	597
Организация и концептуализация	598
Создание многофайловой программы	598
Заголовочные файлы	599
Директории	599
Проекты	600
Межфайловое взаимодействие	600
Взаимодействие исходных файлов	600

Заголовочные файлы	605
Пространства имен	609
Класс сверхбольших чисел	613
Числа как строки	613
Описатель класса	614
Методы	615
Прикладная программа	617
Моделирование высотного лифта	619
Работа программы ELEV	620
Проектирование системы	621
Листинг программы ELEV.	623
Диаграмма состояний для программы ELEV	634
Резюме	635
Вопросы	636
Проекты	638
Глава 14. Шаблоны и исключения	640
Шаблоны функций	640
Шаблон простой функции	642
Шаблоны функций с несколькими аргументами	644
Шаблоны классов	647
Контекстозависимое имя класса	651
Создание класса связанных списков с помощью шаблонов	653
Хранение пользовательских типов	655
UML и шаблоны	658
Исключения	659
Для чего нужны исключения	660
Синтаксис исключений	661
Простой пример исключения	662
Множественные исключения	666
Исключения и класс Distance	668
Исключения с аргументами	670
Класс bad_alloc	673
Размышления об исключениях	674
Резюме	675
Вопросы	676
Упражнения	678
Глава 15. Стандартная библиотека шаблонов (STL)	681
Введение в STL	682
Контейнеры	682
Алгоритмы	687
Итераторы	688
Возможные проблемы с STL	689
Алгоритмы	690
Алгоритм find ().	690
Алгоритм count ().	691
Алгоритм sort().	692
Алгоритм search().	692

Алгоритм merge().	693
Функциональные объекты	694
Пользовательские функции вместо функциональных объектов	695
Добавление _if к аргументам	696
Алгоритм for_each ()	697
Алгоритм transform()	697
Последовательные контейнеры	698
Векторы	699
Списки	702
Итераторы	706
Итераторы как интеллектуальные указатели	706
Итераторы в качестве интерфейса	708
Соответствие алгоритмов контейнерам	710
Работа с итераторами	713
Специализированные итераторы	717
Адаптеры итераторов	717
Потоковые итераторы	720
Ассоциативные контейнеры	724
Множества и мультимножества	725
Отображения и мультиотображения	729
Ассоциативный массив	730
Хранение пользовательских объектов	731
Множество объектов person	732
Список объектов класса person	735
Функциональные объекты	738
Предопределенные функциональные объекты	739
Создание собственных функциональных объектов	741
Функциональные объекты и поведение контейнеров.	746
Резюме	746
Вопросы	747
Упражнения	749

Глава 16. Разработка объектно-ориентированного ПО . . 752

Эволюция процесса создания программного обеспечения	752
Процесс просиживания штанов	753
Каскадный процесс.	753
Объектно-ориентированное программирование	753
Современные подходы	754
Моделирование вариантов использования.	755
Действующие субъекты	755
Варианты использования.	756
Сценарии.	756
Диаграммы вариантов использования	757
Описания вариантов использования	758
От вариантов использования к классам	758
Предметная область программирования.	759
Рукописные формы.	760
Допущения	762
Программа LANDLORD: стадия развития	762
Действующие субъекты	762

Варианты использования	762
Описание вариантов использования	763
Сценарии	765
Диаграммы действий UML	765
От вариантов использования к классам	766
Список существительных	766
Уточнение списка	767
Определение атрибутов	768
От глаголов к сообщениям	768
Диаграмма классов	770
Диаграммы последовательностей	770
Написание кода	774
Заголовочный файл	775
Исходные .сpp файлы	780
Вынужденные упрощения	789
Взаимодействие с программой	789
Заключение	791
Резюме	791
Вопросы	792
Проекты	794

Приложение А. Таблица ASCII 796

Приложение Б. Таблица приоритетов операций C++ . . . 803

Таблица приоритетов операций	803
Зарезервированные слова	803

Приложение В. Microsoft Visual C++ 806

Элементы экрана	806
Однофайловые программы	807
Компоновка существующего файла	807
Создание нового файла	808
Ошибки	808
Информация о типах в процессе исполнения (RTTI)	808
Многофайловые программы	809
Проекты и рабочие области	809
Работа над проектом	809
Сохранение, закрытие и открытие проектов	810
Компиляция и компоновка	811
Программы с консольной графикой	811
Отладка программ	811
Пошаговая трассировка	812
Просмотр переменных	812
Пошаговая трассировка функций	812
Точки останова	813

Приложение Г. Borland C++ Builder 814

Запуск примеров в C++ Builder	815
Очистка экрана	815

Создание нового проекта	815
Задание имени и сохранение проекта	817
Работа с существующими файлами	817
Компиляция, связывание и запуск программ	818
Запуск программы в C++ Builder	818
Запуск программы в MS DOS	818
Предварительно скомпилированные заголовочные файлы.	818
Закрытие и открытие проектов	818
Добавление заголовочного файла к проекту	819
Создание нового заголовочного файла	819
Редактирование заголовочного файла	819
Определение местонахождения заголовочного файла	819
Проекты с несколькими исходными файлами	820
Создание дополнительных исходных файлов	820
Добавление существующих исходных файлов	820
Менеджер проектов	821
Программы с консольной графикой	821
Отладка	822
Пошаговый прогон	822
Просмотр переменных	822
Пошаговая трассировка функций	822
Точки останова	823

Приложение Д. Упрощенный вариант консольной графики 824

Использование подпрограмм библиотеки консольной графики	825
Функции библиотеки консольной графики	825
Реализация функций консольной графики	827
Компиляторы Microsoft	827
Компиляторы Borland.	828
Листинги исходных кодов	828

Приложение Е. Алгоритмы и методы STL 836

Алгоритмы	836
Методы	843
Итераторы	845

Приложение Ж. Ответы и решения 847

Глава 1	847
Глава 2	847
Глава 3	849
Глава 4	853
Глава 5	855
Глава 6	859
Глава 7	862
Глава 8	866
Глава 9	871
Глава 10	876
Глава 11	881

Глава 12	886
Глава 13	889
Глава 14	890
Глава 15	894
Глава 16	898

Приложение 3. Библиография 899

Углубленное изучение C++	899
Основополагающие документы	900
UML	900
История C++	901
И другое	901

Алфавитный указатель 902

Глава 1

Общие сведения

- ◆ Зачем нужно объектно-ориентированное программирование
- ◆ Характеристики объектно-ориентированных языков
- ◆ C++ и C
- ◆ Изучение основ
- ◆ Универсальный язык моделирования (UML)

Изучив эту книгу, вы получите основные навыки создания программ на языке C++, поддерживающем *объектно-ориентированное программирование* (ООП). Для чего нужно ООП? Каковы преимущества ООП перед такими традиционными языками программирования, как Pascal, C или BASIC? Что является основой ООП? В ООП существует две ключевые концепции — *объекты* и *классы*. Каков смысл этих двух терминов? Как связаны между собой языки C и C++?

Эта глава посвящена рассмотрению перечисленных вопросов, а также обзору средств языка, о которых пойдет речь в других главах книги. Не беспокойтесь, если материал, изложенный в этой главе, покажется вам чересчур абстрактным. Все методы и концепции, которые мы упомянем, будут подробно рассмотрены в последующих главах книги.

Для чего нужно объектно-ориентированное программирование?

Развитие объектно-ориентированного метода обусловлено ограниченностью других методов программирования, разработанных ранее. Чтобы лучше понять и оценить значение ООП, необходимо разобраться, в чем состоит эта ограниченность и каким образом она проявляется в традиционных языках программирования.

Процедурные языки

C, Pascal, FORTRAN и другие сходные с ними языки программирования относятся к категории *процедурных языков*. Каждый оператор такого языка является ука-

занием компьютеру совершить некоторое действие, например принять данные от пользователя, произвести с ними определенные действия и вывести результат этих действий на экран. Программы, написанные на процедурных языках, представляют собой последовательности инструкций.

Для небольших программ не требуется дополнительной внутренней организации (часто называемой термином *парадигма*). Программист создает перечень инструкций, а компьютер выполняет действия, соответствующие этим инструкциям.

Деление на функции

Когда размер программы велик, список команд становится слишком громоздким. Очень небольшое число программистов способно удерживать в голове более 500 строк программного кода, если этот код не разделен на более мелкие логические части. *Функция* является средством, облегчающим восприятие при чтении текста программы (термин *функция* употребляется в языках С и С++; в других языках программирования это же понятие называют подпрограммой или процедурой). Программа, построенная на основе процедурного метода, разделена на функции, каждая из которых в идеальном случае выполняет некоторую законченную последовательность действий и имеет явно выраженные связи с другими функциями программы.

Можно развить идею разбиения программы на функции, объединив несколько функций в *модуль* (зачастую модуль представляет собой отдельный файл). При этом сохраняется процедурный принцип: программа делится на несколько компонентов, каждый из которых представляет собой набор инструкций.

Деление программы на функции и модули является основой структурного программирования. Структурное программирование представляет собой нечто не вполне определенное, однако в течение нескольких десятков лет, пока не была разработана концепция объектно-ориентированного программирования, оно оставалось важным способом организации программ.

Недостатки структурного программирования

В непрекращающемся процессе роста и усложнения программ стали постепенно выявляться недостатки структурного подхода к программированию. Возможно, вам приходилось слышать «страшные истории» о том, как происходит работа над программным проектом, или даже самим участвовать в создании такого проекта: задача оказывается сложнее, чем казалось, сроки сдачи проекта переносятся. Все новые и новые программисты привлекаются для работы, что резко увеличивает расходы. Окончание работы вновь переносится, и в результате проект терпит крах.

Проанализировав причины столь печальной судьбы многих проектов, можно прийти к выводу о недостаточной мощи структурного программирования: как бы эффективно ни применялся структурный подход, он не позволяет в достаточной степени упростить большие сложные программы.

В чем же недостаток процедурно-ориентированных языков? Существует две основные проблемы. Первая заключается в неограниченности доступа функций к глобальным данным. Вторая состоит в том, что разделение данных и функций, являющееся основой структурного подхода, плохо отображает картину реального мира.

Давайте рассмотрим эти недостатки на примере программы складского учета. В такой программе глобальными данными являются записи в учетной книге. Различные функции будут получать доступ к этим данным для выполнения операций создания новой записи, вывода записи на экран, изменения существующей записи и т. д.

Неконтролируемый доступ к данным

В процедурной программе, написанной, к примеру, на языке С, существует два типа данных. *Локальные данные* находятся внутри функции и предназначены для использования исключительно этой функцией. Например, в программе складского учета функция, осуществляющая вывод записи на экран, может использовать локальные данные для хранения информации о выводимой записи. Локальные данные функции недоступны никому, кроме самой функции, и не могут быть изменены другими функциями.

Если существует необходимость совместного использования одних и тех же данных несколькими функциями, то данные должны быть объявлены как *глобальные*. Это, как правило, касается тех данных программы, которые являются наиболее важными. Примером здесь может служить уже упомянутая учетная книга. Любая функция имеет доступ к глобальным данным (мы не рассматриваем случай группирования функций в модули). Схема, иллюстрирующая концепцию локальных и глобальных данных, приведена на рис. 1.1.



Рис. 1.1. Глобальные и локальные переменные

Большие программы обычно содержат множество функций и глобальных переменных. Проблема процедурного подхода заключается в том, что число возможных связей между глобальными переменными и функциями может быть очень велико, как показано на рис. 1.2.

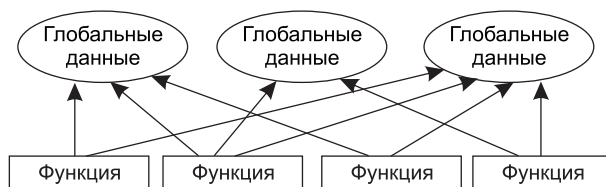


Рис. 1.2. Процедурный подход

Большое число связей между функциями и данными, в свою очередь, также порождает несколько проблем. Во-первых, усложняется структура программы. Во-вторых, в программу становится трудно вносить изменения. Изменение структуры глобальных данных может потребовать переписывания всех функций, работающих с этими данными. Например, если разработчик программы складского учета решит сделать код продукта не 5-значным, а 12-значным, то будет необходимо изменить соответствующий тип данных с `short` на `long`. Это означает, что во все функции, оперирующие кодом продукта, должны быть внесены изменения, позволяющие обрабатывать данные типа `long`. Можно привести аналогичный бытовой пример, когда в супермаркете изменяется расположение отделов, и покупателям приходится соответствующим образом менять свой привычный путь от одного отдела к другому.

Когда изменения вносятся в глобальные данные больших программ, бывает непросто быстро определить, какие функции необходимо скорректировать. Даже в том случае, когда это удастся сделать, из-за многочисленных связей между функциями и данными исправленные функции начинают некорректно работать с другими глобальными данными. Таким образом, любое изменение влечет за собой далеко идущие последствия.

Моделирование реального мира

Вторая, более важная, проблема процедурного подхода заключается в том, что отделение данных от функций оказывается малопригодным для отображения картины реального мира. В реальном мире нам приходится иметь дело с физическими объектами, такими, например, как люди или машины. Эти объекты нельзя отнести ни к данным, ни к функциям, поскольку реальные вещи представляют собой совокупность *свойств* и *поведения*.

Свойства

Примерами свойств (иногда называемых характеристиками) для людей могут являться цвет глаз или место работы; для машин — мощность двигателя и количество дверей. Таким образом, свойства объектов равносильны данным в программах: они имеют определенное значение, например *голубой* для цвета глаз или *4* для количества дверей автомобиля.

Поведение

Поведение — это некоторая реакция объекта в ответ на внешнее воздействие. Например, ваш босс в ответ на просьбу о повышении может дать ответ «да» или

«нет». Если вы нажмете на тормоз автомобиля, это повлечет за собой его остановку. Ответ и остановка являются примерами поведения. Поведение сходно с функцией: вы вызываете функцию, чтобы совершить какое-либо действие (например, вывести на экран учетную запись), и функция совершает это действие.

Таким образом, ни отдельно взятые данные, ни отдельно взятые функции не способны адекватно отобразить объекты реального мира.

Объектно-ориентированный подход

Основопологающей идеей объектно-ориентированного подхода является объединение *данных* и *действий, производимых над этими данными*, в единое целое, которое называется *объектом*.

Функции объекта, называемые в C++ *методами* или *функциями-членами*, обычно предназначены для доступа к данным объекта. Если необходимо считать какие-либо данные объекта, нужно вызвать соответствующий метод, который выполнит считывание и возвратит требуемое значение. Прямой доступ к данным невозможен. Данные сокрыты от внешнего воздействия, что защищает их от случайного изменения. Говорят, что данные и методы *инкапсулированы*. Термины *сокрытие* и *инкапсуляция* данных являются ключевыми в описании объектно-ориентированных языков.

Если необходимо изменить данные объекта, то, очевидно, это действие также будет возложено на методы объекта. Никакие другие функции не могут изменять данные класса. Такой подход облегчает написание, отладку и использование программы.

Типичная программа на языке C++ состоит из совокупности объектов, взаимодействующих между собой посредством вызова методов друг друга. Структура программы на C++ приводится на рис. 1.3.

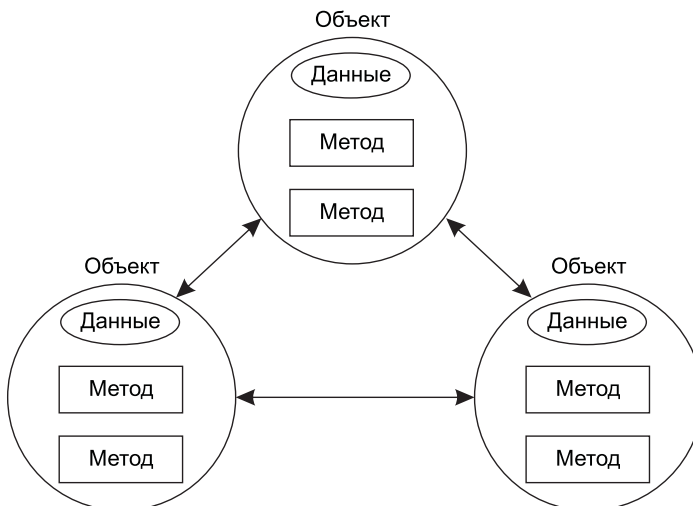


Рис. 1.3. Объектно-ориентированный подход