Содержание

[ЧАСТЬ I]

Шаблоны, требующие знания переменных, их типов и арифметических операторов

Часть I содержит шаблоны программирования, которые требуют понимания типов данных, переменных и идентификаторов, объявления переменных и базового понимания структуры памяти, оператора присваивания и его влияния на память, а также арифметических операторов. Многие шаблоны в этой части книги широко используют целочисленную арифметику.

В частности, здесь содержатся следующие шаблоны программирования.

Обновление. Решение задачи обновления переменной.

Обмен. Решение проблемы перестановки содержимого двух переменных (одного и того же типа).

Манипуляции с цифрами. Решение задач извлечения и удаления цифр целого числа (как с правой, так и с левой стороны).

Арифметика на числовой окружности. Решение проблемы выполнения базовых арифметических операций над величинами, которые повторяются (например, дни недели, месяцы года).

Усечение. Решение проблемы усечения целого числа до определенной цифры (т. е. до степени 10).

Шаблоны, описанные в этой части книги, встречаются так часто, что стали «второй натурой» опытных программистов, что раздражает начинающих программистов, которым приходится думать о них каждый раз, когда они их используют. Само по себе это наблюдение свидетельствует о важности изучения как этих шаблонов, так и тех, которые описаны в других частях этой книги.

[1]

ОБНОВЛЕНИЕ

Алгоритм любой сложности широко использует арифметические операторы при вычислении какого-то значения, независимо от того, выполняется ли он вручную или реализован в виде компьютерной программы и выполняется на компьютере. Однако эти два способа вычисления имеют очень важное и существенное отличие. Вычисления, выполняемые вручную, как правило, продвигаются вниз по странице, используя по мере выполнения все больше и больше памяти (т. е. площади листа бумаги). Программы же обычно объявляют небольшое количество переменных, присваивают этим переменным значения, а затем обновляют эти присвоенные значения. В этой главе рассматриваются различные способы обновления переменных.

Описание задачи

Предположим, вы пишете программу, которая отслеживает возраст вашего близкого родственника. Вы могли бы объявить переменную целочисленного типа с именем `age` и присвоить этой переменной текущий возраст вашего родственника.

Теперь предположим, что вам нужно произвести вычисления с учетом возраста этого родственника в следующем году. Вы точно знаете, что его возраст будет на единицу больше, чем сейчас. Но следует ли создать другую переменную для этого значения или необходимо просто обновить значение `age`? В некоторых ситуациях вам нужно сделать первое, но предположим, все, что вам нужно, это обновить существующую переменную. Оказывается, существует множество различных способов, которые вы можете использовать.

Обзор

Один из подходов к обновлению, который вы, возможно, уже видели, включает операторы инкремента и декремента, которые увеличивают/уменьшают свои операнды на единицу. Итак, например, вы вероятно видели что-то вроде следующего:

```
let age = 0; // Возраст равен 0 при рождении
age++;       // Увеличим на 1 каждый год
```

[ЧАСТЬ IV]

Шаблоны, требующие углубленных знаний о массивах

Часть IV содержит шаблоны программирования, которые требуют более глубокого понимания массивов и понимания массивов массивов (иногда называемых многомерными массивами). В частности, эта часть книги содержит следующие шаблоны программирования.

Подмассивы. Решение задач, в которых вычисления необходимо выполнять для всех элементов массива, расположенных между двумя индексами.

Окрестности. Решение задач, в которых вычисления необходимо выполнять для всех элементов массива, находящихся «в окрестности» определенного индекса.

Оба этих шаблона предполагают выполнение вычислений над подмножеством элементов массива. Они различаются способом определения подмножества.

[19]

ПОДМАССИВЫ

Описание задачи

Большинство примеров с массивами, которые вы видели, включают в себя перебор всех элементов. Однако во многих ситуациях вам нужно перебрать только некоторые элементы массива. Поскольку в этой книге вы сталкивались в основном с примерами, в которых это не так, вы, возможно, могли начать использовать шаблон, который усложняет решение поставленной задачи (поэтому его иногда называют *антишаблоном*).

Обзор

Предположим, вас попросили написать функцию, которой передается массив целочисленных значений, а возвращается общая сумма элементов массива. Вероятно, вы бы использовали аккумулятор (см. главу 13):

```
function total(data) {
    let result = 0;

    for (let i = 0; i < data.length; ++i) {
        result += data[i];
    }
    return result; }
```

Эта функция предполагает, что количество итераций определено, и использует цикл `for`.

Проблема этой реализации в том, что она не позволяет найти сумму подмножества элементов. Например, если индексы представляют месяцы, а элементы представляют данные о продажах, то вам может потребоваться найти общий объем продаж только за второй квартал (т. е. апрель, май и июнь; номера месяцев 3, 4, 5 отсчитываются от 0).

Ход решения

Очевидно, что вам нужно добавить в функцию формальные параметры, описывающие интересующее подмножество. Например, вы можете передать еще один массив, содержащий рассматриваемые индексы. Или вы можете передать конформный массив логических

[ЧАСТЬ V]

Шаблоны, требующие знания строковых объектов

Часть V содержит шаблоны программирования, требующие понимания строковых объектов. Эта часть книги содержит следующие шаблоны программирования.

Центрирование. Решение проблем, связанных с центрированием содержимого (разных видов) в контейнерах (разных видов).

Разделение строк. Решение задач, аналогичных задаче расстановки запятых между словами в списке.

Динамическое форматирование. Решение проблем, требующих динамического определения формата строки (т. е. во время выполнения, а не во время компиляции).

Множественность. Решение проблемы образования как правильных, так и неправильных форм множественного числа.

Первые три шаблона в этой части книги используют аккумулятор и некоторую другую логику для решения связанных с ними задач. Модель множественности на самом деле представляет собой не что иное, как умелое использование методов.

[21]

ЦЕНТРИРОВАНИЕ

Многим приложениям необходимо центрировать содержимое (какого-либо вида) внутри контейнера (какого-либо вида). Хотя содержимое и контейнеры могут существенно различаться, шаблон, используемый для центрирования, весьма единообразен.

Описание задачи

Предположим, у вас есть текст, который нужно отобразить на консоли, выровняв его по центру строки. Поскольку консоль (обычно) использует моноширинный шрифт, ширина каждого символа (измеряемая в пикселях) одинакова. В результате и ширину текста, и ширину строки можно измерять в символах. Ваша задача — определить столбец строки, в котором должен находиться первый символ текста.

Обзор

Текст в этом примере будет представлен как строковый объект. Итак, вы можете определить количество символов в нем. Но у вас нет возможности указать столбец дисплея, в который будет помещаться текст при выводе на консоль. Следовательно, вам нужно создать или вывести строку, заполненную слева соответствующим количеством пробелов, что можно сделать с помощью аккумулятора (см. главу 13). Единственной проблемой, которая остается, является определение соответствующего количества пробелов.

Ход решения

Предположим, что строка состоит из девяти символов и отсчитывается от 0 (т. е. первый символ находится в позиции 0). Тогда вы знаете, что средний символ в строке имеет индекс 4 (т. е. 9 разделить на 2 с использованием целочисленного деления), поскольку слева от индекса 4 находятся четыре символа, и справа от него — четыре символа.

Предположим далее, что текст имеет ширину пять символов и нумерация также начинается с 0. Вы знаете, что средний символ текста имеет индекс 2 (т. е. 5 разделить целочисленно на 2), поскольку слева от индекса 2 находятся два символа, и справа от него — два символа.

Итак, чтобы центрировать текст в строке, вам нужно, чтобы символ 2 текста находился в позиции 4 строки. Это означает, что символ 0 текста должен находиться в позиции 2 строки.

Шаблон

Шаблон центрирования — не что иное, как обобщение этого примера. Во-первых, вместо текста следует рассматривать более обобщенное содержание. Во-вторых, вместо строки следует думать о контейнере в общем смысле. И у содержимого, и у контейнера есть *размер*, обобщающий понятие ширины в примере, и *ссылка*, обобщающая понятие начального символа.

Задача центрирования состоит в том, чтобы найти ссылку на содержимое, учитывая ссылку и объем контейнера, а также размер содержимого. Обозначим контейнер — C, содержимое — c, верхние индексы R, E и M — обозначают ссылку, размер и среднюю точку соответственно (для каждого измерения), шаблон центрирования включает три этапа.

Сначала вам нужно вычислить среднюю точку контейнера (в примере ссылка равна 0, а размер — 9). Вы можете рассчитать место расположения середины следующим образом:

$$C^M = C^R + (C^E / 2)$$

Далее вам необходимо вычислить среднюю точку содержимого (ширина которого в примере — 5). Это можно сделать следующим образом:

$$c^M = (c^E / 2)$$

Наконец, вам нужно рассчитать начальную точку (ссылку) для содержимого, вычитая координату средней точки содержимого из координаты средней точки контейнера:

$$c^R = C^M - c^M$$

В одном измерении (т. е. когда ссылки и размеры могут быть представлены одним числом), как в текстовом примере, этот алгоритм можно реализовать следующим образом:

```
function center(containerReference, containerExtent,
                contentExtent) {
    const containerMidpoint = containerReference +
                              containerExtent / 2.0;
```

[28]

КОНТРОЛЬНЫЕ СПИСКИ

Во многих аспектах жизни, как личной, так и профессиональной, необходимо определить, был ли удовлетворен/выполнен набор критериев (например, достигнуты ли цели, пройдены ли курсы). Один из распространенных способов сделать это — использовать контрольный список.

Описание задачи

Предположим, вы собираетесь в отпуск. Вероятно, у вас есть несколько вещей, которые вы хотите не забыть взять с собой (например, рубашки, носки, брюки и юбки). Итак, вы решаете написать программу, которая поможет вам ничего не забыть. Однако, в отличие от ситуаций, рассмотренных в главе 10 о битовых флагах, контрольный список будет предоставляться программе динамически во время выполнения (т. е. он не будет известен в момент написания и компиляции программы).

Обзор

Чтобы повысить гибкость программы, вы решаете представить критерии, которые должны быть удовлетворены/выполнены, в виде массива строковых переменных с именем `checklist`, и заполняете этот массив перед тем, как начать работать над задачами. Затем, по мере выполнения задачи, вы вводите ее в другой массив строковых переменных, который называется `accomplished`. Каждый раз, когда вы выполняете задачу, вы хотите иметь возможность определить, выполнили вы работу или нет (например, выполнили ли вы все задачи из контрольного списка).

Конечно, легко сравнить один элемент из списка `checklist` с одним элементом из списка `accomplished`, используя метод `equals()` в классе строковых переменных. Однако само по себе это не решает проблему определения того, закончили вы работу или нет. Очевидно, что метод `equals()` должен вызываться итеративно.

Ход решения

Вы можете определить, был ли выполнен тот или иной элемент из списка `checklist`, сравнивая его с каждым элементом из списка `accomplished`. Например, вы можете определить, выполнен ли элемент `index` из списка `checklist` следующим образом: