Министерство образования и науки Российской Федерации

ГОСУДАРСТВЕННОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ
РОССИЙСКИЙ ГОСУДАРСТВЕННЫЙ ГИДРОМЕТЕОРОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ

В.В. КОВАЛЕНКО, Е.В. ГАЙДУКОВА

ПРАКТИКУМ ПО ДИСЦИПЛИНЕ «МОДЕЛИРОВАНИЕ ГИДРОЛОГИЧЕСКИХ ПРОЦЕССОВ. ЧАСТЬ I. ДИНАМИЧЕСКИЕ МОДЕЛИ»

(на базе языка С++)

Допущено Учебно-методическим объединением по образованию в области гидрометеорологии в качестве учебного пособия для студентов высших учебных заведений, обучающихся по специальности «Гидрология»

Санкт-Петербург
2010

УДК [556:004.434:004.94].072(075.8)
ББК 26.22
К56

Коваленко В. В., Гайдукова Е. В. Практикум по дисциплине «Моделирование гидрологических процессов. Часть I. Динамические модели» (на базе языка С++). Учебное пособие. – СПб.: изд. РГГМУ, 2010. – 147 с.

ISBN 978-5-86813-287-2

Рецензент: д-р физ.-мат. наук С. А. Кондратьев (зам. директора Института озероведения РАН).

В учебном пособии рассматриваются примеры моделирования гидрологических процессов. Оно охватывает первую часть программы дисциплины «Моделирование гидрологических процессов», связанную с использованием динамических моделей. Приводятся примеры численного решения моделей гидрологического цикла как с сосредоточенными, так и с распределенными параметрами. Рассматриваемые примеры адаптированы к уровню студентов III курса. Особенностью «Практикума» является то, что студентам предлагается вначале решить примеры «вручную», а затем, после знакомства с элементами языка С++, путем составления соответствующих программ.

Предназначена студентам-гидрологам высших учебных заведений, но материал доступен и учащимся техникумов соответствующего профиля.

Учебное пособие одобрено Ученым советом РГГМУ (протокол № 3 от 23.11.2010 г.)

Kovalenko V. V., Gaidukova E. V. Practical work at the discipline «Modeling of hydrological processes. Part I. Dynamic models» (on the basis of language C++). The manual. – St. Petersburg, RSHU Publishers, 2010. – 147 pp.

Reviewers: the doctor of physical and mathematical sciences S. A. Kondratyev (the deputy director of Institute of lake study the Russian Academy of Sciences).

In the manual the examples of modeling of hydrological processes are considered. It covers the first part of the program of a discipline «Modeling of hydrological processes», connected with use of dynamic models. The examples of the numerical decision of models of a hydrological cycle both with concentrated, and with the distributed parameters are resulted. The considered examples are adapted to a level of the students of third rate. Feature of «practical work» is that the students are offered in the beginning to solve examples «in hand-operated», and then, after acquaintance to elements of language C++, by drawing up of the appropriate programs.

Is intended to the students-hydrologists of higher educational institutions, but the material is accessible also to pupil of technical schools of the appropriate structure.

ISBN 978-5-86813-287-2

Введение

Практикум состоит из трех частей, что соответствует действующей программе данного курса (динамические модели, стохастические модели, частично инфинитное моделирование) и учебнику [3]. В настоящей (первой) части рассматриваются примеры использования динамических моделей гидрологического цикла. Особенностью практикума является то, что он построен на базе профессионального языка программирования С++. По существу, в трех частях охвачены все основные элементы этого языка, а также его расширение – С++ Builder.

Один из авторов, обращаясь к своим студенческим годам, вспоминает первые попытки Министерства образования внедрить в учебные планы технических ВУЗов элементы программирования. Программы, написанные чуть ли не в машинных кодах, перфокарты и очереди, чтобы «пробиться» с этими перфокартами к машине. Лучшего способа, чтобы вызвать отвращение к программированию и не придумать.

Сейчас все изменилось, но многие ли студенты-гидрологи горят желанием научиться программировать? Отсутствие системного (начиная с первого и кончая последним годом обучения) подхода к освоению «компьютерной грамотности» делает свое дело: зачем себя насиловать, если можно «проскочить» и так. Специальные дисциплины преподаются таким образом, что можно обойтись стандартным набором коммерческих программ (типа Excel или MatLab). И тем не менее все настоятельнее пробивается потребность в знании программирования.

Авторы не профессиональные программисты, но в кое-чем все-таки разбираются. Они знают, как можно научить программировать на С++. Разумеется, ни за какой «21 день» освоить С++ невозможно. Это долгий, трудный (а иногда и нудный) путь, и надо себя настроить на то, чтобы выдержать его на протяжении 2^x–3^x лет.

За основу данного пособия взяты несколько известных книг для изучения С++ [4, 5, 6, 8, 9], которые по мере сил были «огидрологичены». Последнее касалось в основном примеров программ, так как ни синтаксис, ни основные конструкции языка никакая гидрология изменить не может. Первые две части практикума со-

держат достаточно полное изложение основ языка, удовлетворяющего аппаратно независимому стандарту ANSI (его версии ISO/IEC 14882-1998; современная версия – ISO/IEC 14882-2003). В третьей части рассматривается среда визуального программирования BORLAND C++ BULDER.

Современное моделирование немыслимо без применения численных методов. В данном пособии использованы наиболее простые из них, допускающие «ручную» реализацию. Только в некоторых случаях упоминаются более сложные (и то в сравнении с простыми, чтобы мотивировать студентов к использованию программных средств реализации моделей). В учебном плане РГГМУ предусмотрена отдельная дисциплина «Численные методы в гидрологии», где подробно рассматриваются особенности тех или иных конечно-разностных аппроксимаций дифференциальных уравнений.

Материал пособия прошел практическую апробацию, которая показала, что на каждую из 3-х частей достаточно одного семестра (четыре часа занятий в неделю). Важными элементами обучения являются семинары и выполнение практических заданий, закладывающих основу навыков программирования. На семинарах рассматриваются практические вопросы взаимодействия C++ с коммерческими программами, а также намечающиеся тупики при попытке применения стандартного C++ к программированию задач частично инфинитной гидрологии (см. [1, 2]), связанных с моделированием и прогнозированием статистически неустойчивых развивающихся гидрологических систем.

Подготовка учебного пособия инициализирована грантом Министерства образования и науки РФ по проекту № 2.1.1/3355 «Создание диагностических и прогностических моделей развития процессов катастрофического формирования многолетнего речного стока» аналитической ведомственной целевой программы «Развитие научного потенциала высшей школы (2009–2010 годы)», мероприятие: 2, Проведение фундаментальных исследований в области естественных, технических и гуманитарных наук. Научно-методическое обеспечение развития инфраструктуры вузовской науки; раздел: 2.1, Проведение фундаментальных исследований в области естественных, технических и гуманитарных наук; подраздел: 2.1.1, Проведение фундаментальных исследований в области естественных наук.

1. Рассматриваемые примеры и их «ручная» реализация

1.1. Модель склонового стока с сосредоточенными параметрами

1.1.1. Одно- и двухъемкостная модели

Модель склонового стока с сосредоточенными параметрами имеет следующий вид:

$$\frac{dQ}{dt} = -\frac{1}{k\tau}Q + \frac{\dot{X}}{\tau}, \qquad (1.1)$$

где Q – расход (модуль, слой) стока; $\dot{X}$ – интенсивность осадков; k – коэффициент стока; τ – время релаксации (добегания).

Для решения модели (1.1) требуется знание задаваемых параметров k и τ. Методика их оптимизации заключается в следующем.

1. Рассчитывается допустимая погрешность:

$$\delta_{доп} = \pm 0{,}674\sigma_{\Delta}, \qquad (1.2)$$

где σ_{Δ} – среднее квадратическое отклонение изменения прогнозируемой величины за период заблаговременности прогноза от среднего значения этого изменения:

$$\sigma_{\Delta} = \sqrt{\frac{\sum_{i=1}^{n}(\Delta_i - \overline{\Delta})^2}{n-1}} \qquad (1.3)$$

(здесь Δ_i – изменение прогнозируемой величины за период заблаговременности прогноза; $\overline{\Delta}$ – среднее значение этих изменений; n – число изменений).

Заблаговременность краткосрочных прогнозов времени наступления гидрологических явлений определяется периодом от даты выпуска прогноза до указанной в прогнозе даты.

2. Оценивается оправдываемость отдельного прогноза. Прогноз считается оправдавшимся, если абсолютная величина его погрешности меньше или равна допустимой.

3. Оценивается эффективность методики. Мерой точности методики прогнозирования является средняя квадратическая погрешность поверочных прогнозов, вычисляемая по формуле:

$$S = \sqrt{\frac{\sum_{i=1}^{n}\left(y_i - y_i'\right)^2}{n-m}}, \qquad (1.4)$$

где y_i и y_i' – соответственно фактическое и предсказанное значения; n – число членов ряда; m – число степеней свободы, равное числу постоянных в прогностическом уравнении.

За критерий применимости и качества методики принимается отношение S/σ.

Погрешности определения S и σ зависят и от числа членов ряда n, поэтому учитываются следующие условия применимости методик прогнозирования:

при $n \le 15$ $\quad S/\sigma \le 0.70$*,
при $15 < n < 25$ $\quad S/\sigma \le 0.75$,
при $n \ge 25$ $\quad S/\sigma \le 0.80$.

4. Определяется обеспеченность методики:

$$P = \frac{n'}{n}100, \qquad (1.5)$$

где n' – число оправдавшихся прогнозов.

5. За оптимальные значения коэффициента стока и времени добегания принимаются те значения, с которыми при поверочном прогнозе получены минимальное соотношение S/σ и максимальная величина P. (Методика оптимизации сводится к варьированию значений k, τ и реализована программно в п. 3.)

Уравнение (1.1) описывает процесс в некой «емкости» (речном бассейне), заполняемой водой ($\dot{X}$), часть которой «теряется»

* Здесь и ниже в качестве разделителя в десятичных дробях используется точка.

(потери учитываются коэффициентом стока k), а другая часть с каким-то опаздываниям (учитываемым времени добегания τ) «сбрасывается» через замыкающий створ. Однако бассейн более сложное образование – это и русловая сеть, и почво-грунты, и, возможно, подземные резервуары (например, карстовые образования). Поэтому расширим модель (1.1) до двухъемкостной.

Пусть в формировании стока участвуют два резервуара (рис. 1.1): поверхностный (параметры k и τ_1) и подземный, который в конечном итоге разгружается в реку со временем релаксации τ_2. Балансовое уравнение для верхнего резервуара $dW_1/dt = (\dot{X} - Q/k)$, учитывая, что $W_1 \approx \tau_1 Q_1$, запишем так:

$$dQ_1/dt = (\dot{X} - Q_1/k)/\tau_1. \qquad (1.6)$$

Аналогично для второго резервуара ($dW_2/dt = Q_1 - Q_2$):

$$dQ_2/dt = (Q_1 - Q_2)/\tau_2. \qquad (1.7)$$

Объединяя (1.6) и (1.7), получаем:

$$\tau_2 \frac{d^2Q}{dt^2} + \left(\frac{\tau_2}{k\tau_1} + 1\right)\frac{dQ}{dt} + \frac{1}{k\tau_1}Q = \frac{1}{\tau_1}\dot{X} \qquad (1.8)$$

(при $\tau_2 = 0$ модель сводится к уравнению (1.1)).

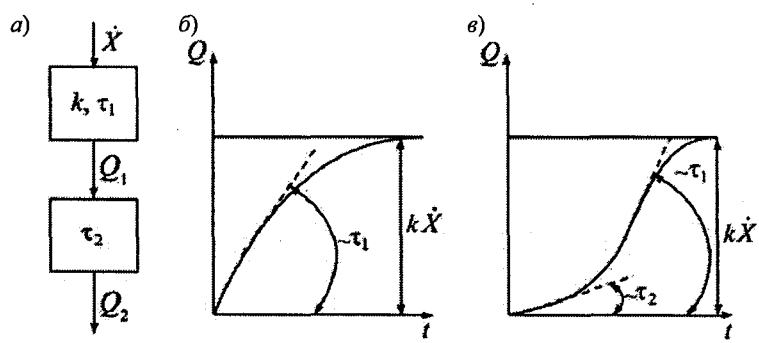


Рис. 1.1. Схема двухъемкостной структуры формирования стока (*а*), реакция одноемкостной (*б*) и двухъемкостной (*в*) моделей на «ступенчатое» воздействие осадков (тангенсы углов наклона пропорциональны временам релаксации τ_1 и τ_2).

1.1.2. Численная реализация моделей методами Эйлера и Рунге-Кутты

Двухъемкостная модель (1.8) может быть приведена к системе двух обыкновенных дифференциальных уравнений первого порядка:

$$d\vec{Q}/dt = f(\vec{Q},t) \qquad (1.9)$$

(здесь $\vec{Q} = (Q_1,\ Q_2)$).

Наиболее распространенным методом решения подобных систем является метод Рунге-Кутты. Он основан на разложении решения (1.9) в ряд Тейлора в окрестности узлов $t = t_n$ $(n = 0,\ 1,\ ...)$

$$\vec{Q}(t_{n+1}) = \vec{Q}(t_n) + \Delta t\vec{Q}'(t_n) + ((\Delta t)^2/2)\vec{Q}''(t_n) + ... \qquad (1.10)$$

(здесь $\Delta t = t_{n+1} - t_n$) и отбрасывании членов соответствующего порядка. Например, чтобы получить метод Рунге-Кутты первого порядка (он же метод Эйлера), надо отбросить в разложении члены второго порядка.

Наиболее востребован метод четвертого порядка, который для одного уравнения (1.1) $dQ/dt = f(Q,t)$ выглядит так:

$$\begin{gathered} Q_{n+1} = Q_n + \frac{\Delta t}{6}(l_1 + 2l_2 + 2l_3 + l_4), \quad n = 0,\ 1,\ ... \\ l_1 = f(Q_n, t_n), \quad l_2 = f(Q_n + l_1/2, \quad t_n + \Delta t/2), \\ l_3 = f(Q_n + l_2/2, \quad t_n + \Delta t/2), \quad l_4 = f(Q_n + l_3, \quad t_n + \Delta t). \end{gathered} \qquad (1.11)$$

Конечно, двухъемкостную модель (1.8) можно аппроксимировать и непосредственно:

$$\tau_2 \frac{Q_{n+1} - 2Q_n + Q_{n-1}}{\Delta t^2} + (\frac{\tau_2}{k\tau_1} + 1)(\frac{Q_{n+1} - Q_n}{\Delta t}) + \frac{1}{k\tau_1} Q_n = \frac{\dot{X}_n}{\tau_1}. \qquad (1.12)$$

С одной стороны, метод Эйлера проще реализовать вручную, хотя, с другой стороны для него существуют жесткие ограничения на значения расчетного шага Δt. Рисунок 1.2 иллюстрирует реакцию обеих моделей на ступенчатое воздействие осадков, а также на возникновение неустойчивости решений при использовании метода Эйлера и Рунге-Кутты четвертого порядка.

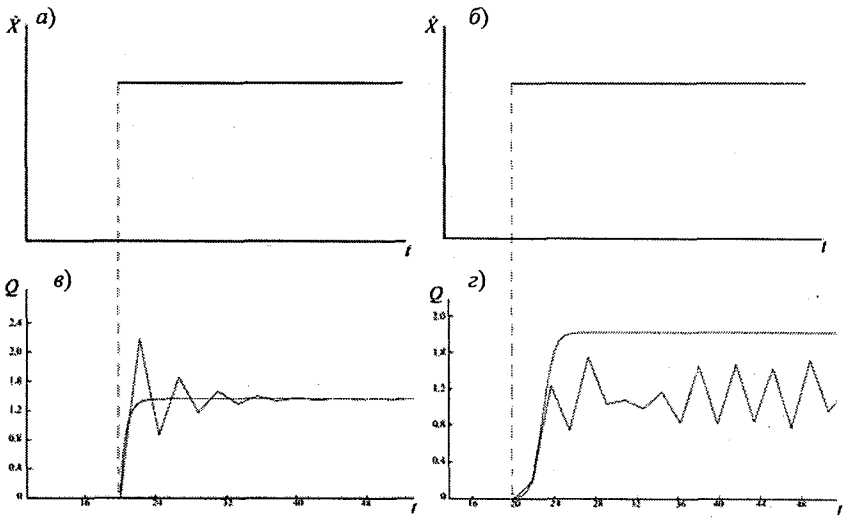


Рис. 1.2. Реакция одноемкостной (*в*) и двухъемкостной (*г*) моделей на ступенчатое изменение осадков (*а*, *б*).
Ломанными линиями показаны варианты решений, возникающие при нарушении условий устойчивости.

1.2. Модель руслового стока с распределенными параметрами

1.2.1. Кинематическая волна

Поверхностный сток воды может быть и склоновым, и русловым. Если для руслового стока на участке реки в качестве основной искомой величины принять площадь живого сечения (F), а в качестве внешних воздействий – боковой приток со склонов q (распределенный или сосредоточенный) или приток (отток) воды в

русло $\pm k$ ($H_р$ – – $H_{гр}$) в результате взаимодействия с грунтовыми водами (k – коэффициент, $H_{гр}$ – уровень грунтовых вод, $H_р$ – уровень воды в русле), т. е. считать внешним воздействием $\varepsilon = f(q, \pm k(H_р - H_{гр})$, то для одномерной идеализации в соответствии с общими правилами построения моделей [3] получим:

$$\partial F / \partial t + \partial Q / \partial x = \varepsilon(x, t), \qquad (1.13)$$

где x – продольная координата.

Уравнение неразрывности (1.13) выражает закон сохранения массы воды на участке реки. Одной из наиболее простых моделей движения воды в русле является модель кинематической волны. Основная ее предпосылка состоит в том, что, несмотря на неустановившийся характер движения воды, описываемой этой моделью, связь расходов воды и уровней (площадей) принимается однозначной:

$$Q = f(F, x). \qquad (1.14)$$

Подставив (1.14) в (1.13), получим модель кинематической волны:

$$\partial F / \partial t + \partial f(F, x) / \partial x = \varepsilon(x, t). \qquad (1.15)$$

Функция $f(F, x)$ должна быть задана, т. е. из предварительных измерений или расчетов должна быть известна связь расходов воды и площадей на рассматриваемом участке русла (она характеризует пропускную способность русла). Решив это уравнение при заданном внешнем воздействии, например боковой приточности, верхнем граничном условии $F = F(x_в, t)$ и начальном условии $F = F(x, t_0)$, можно найти $F = F(x, t)$, а следовательно, и $Q = Q(x, t)$, так как $Q = f(F, x)$.

1.2.2. Пример решения задачи, связанной с распространением по руслу волны попуска

Для решения модели кинематической волны обычно применяется метод сеток. Запишем расчетное уравнение при линейной свя-

зи $Q = aF$ (здесь a – скорость перемещения волны):

$$\partial F / \partial t + a \partial F / \partial x = q . \qquad (1.16)$$

В методе сеток (рис. 1.2) дифференциальное уравнение заменяется конечно-разностным, например:

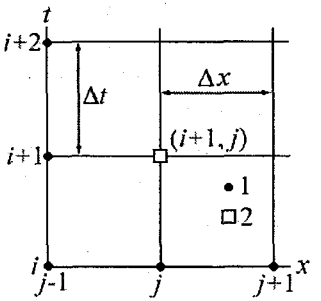


Рис. 1.2. Прямоугольная сетка.
1 – внешние узлы, *2* – внутренний узел.

$$(F_{i+1,\,j} - F_{i,\,j}) / \Delta t + a (F_{i,\,j} - F_{i,\,j-1}) / \Delta x = q_{i,\,j} .$$

Считая, что площади по всей длине участка в начальный момент времени и на границе во все моменты интересующего нас промежутка времени известны (на внешних узлах сетки), находим решение во внутренних узлах по следующему алгоритму:

$$F_{i+1,\,j} = F_{i,\,j} - a(\Delta t / \Delta x)(F_{i,\,j} - F_{i,\,j-1}) + q_{i,\,j} \Delta t . \qquad (1.17)$$

Передвигаясь последовательно от узла к узлу, находим площадь живого сечения во всех узлах сетки.

Для устойчивости решения необходимо соблюдать определенное соотношение шагов по времени Δt и по продольной координате Δx: $(\Delta t / \Delta x) a \le 1$.

Полученная схема явная, поскольку значения сеточной функции в каждом узле верхнего слоя $t = t_{i+1}$ выражаются явно с помощью соотношений (1.17) через ранее найденные ее значения на предыдущем слое. Если производную $\partial F / \partial x$ аппроксимировать не на i-м слое, а на $(i + 1)$-м, то получится неявная схема. Разностное уравнение примет вид:

$$(F_{i+1,\,j} - F_{i,\,j}) / \Delta t + a (F_{i+1,\,j} - F_{i+1,\,j-1}) / \Delta x = q_{i,\,j} . \qquad (1.18)$$

Рассмотрим **пример**.

Задача: рассчитать динамику изменения площади живого сечения русла длиной 5000 м в течение 5000 с.

Исходные данные:

– аналитическое выражение кривой расходов $Q = aF$ ($a = 0.9$ м/с);

– начальное условие $F|_{t_0} = 1$ м^2 $\forall x \in [0,\ 5000]$;

– граничное условие $F|_{x_0} = 0.001t + 1$ $\forall t \in [0,\ 5000]$;

– боковой приток $q(x, t) = q(x = 4000,\ t) = 0.001$ м^2/с.

Проверим согласованность граничных и начальных условий $F(x = 0) = F(t = 0) = 1$ и зададим шаги дискретизации по расстоянию $\Delta x = 1000$ м и по времени $\Delta t = 1000$ с согласно требованию устойчивости решения ($(\Delta t / \Delta x)a \le 1$).

Используя алгоритм

$$F_{i+1,\,j} = F_{i,\,j} - (a\Delta t / \Delta x)(F_{i,\,j} - F_{i,\,j-1}) + q_{i,\,j}\Delta t\,,$$

выполняем вычисления для каждого узла сетки (табл. 1.1).

Приведем вычисления на первых двух шагах:

$F_{i+1,\,j} = F_{i,\,j} - (a\Delta t / \Delta x)(F_{i,\,j} - F_{i,\,j-1}) + q_{i,\,j}\Delta t = 1.00 - (0.9 \cdot 1000/1000) \cdot (1.00 - 1.00) + 0 \cdot 1000 = 1.00$ м^2;

$F_{i+2,\,j} = F_{i+1,\,j} - (a\Delta t / \Delta x)(F_{i+1,\,j} - F_{i+1,\,j-1}) + q_{i+1,\,j}\Delta t = 1.00 - (0.9 \cdot 1000/1000) \cdot (1.00 - 2.00) + 0 \cdot 1000 = 1.90$ м^2.

Таблица 1.1

Значения площади живого сечения в русле

t с						
i+5	5000	6.00	4.89	3.78	2.68	2.77	2.11
i+4	4000	5.00	3.89	2.78	1.73	2.11	2.11
i+3	3000	4.00	2.89	1.81	1.00	2.11	2.08
i+2	2000	3.00	1.90	1.00	1.00	2.10	1.90
i+1	1000	2.00	1.00	1.00	1.00	2.00	1.00
i		1.00	1.00	1.00	1.00	1.00	1.00
		0	1	2	3	4	5
		j–1	j	j+1	j+2	j+3	j+4

x км

По значениям табл. 1.1 можно построить серию графиков, показывающих изменения площади живого сечения по времени и координате (рис. 1.3).

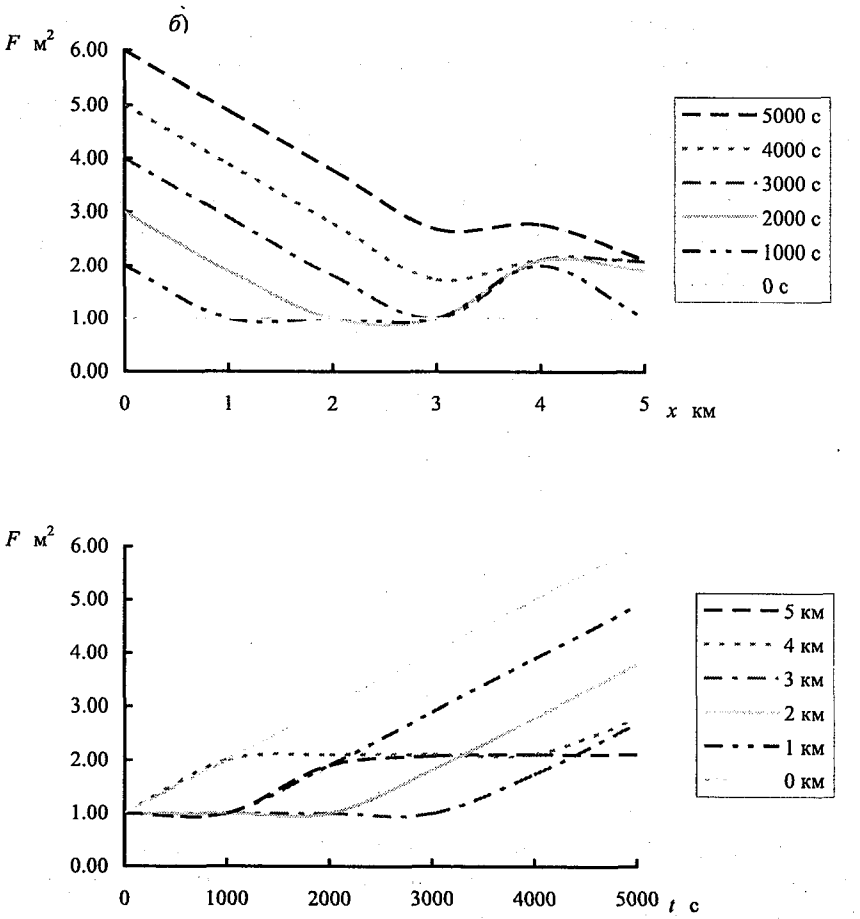


Рис. 1.3. Распределение значений площади живого сечения по времени (*а*) и по длине русла (*б*).

1.3. Системная модель, включающая взаимодействие различных звеньев, участвующих в гидрологическом цикле

1.3.1. Текст задания для самостоятельного решения*

Водохозяйственный объект (задается искусственная ситуация), включающий водоем с двумя втекающими и двумя вытекающими реками (рис. 1.4), на нижней реке на 20-м километре имеет железнодорожный мост с подмостовым отверстием, способным пропускать волны дождевых паводков уровнем не более 3 м над условной плоскостью сравнения. При больших уровнях вода начинает переливаться через железнодорожную насыпь, что приводит к аварии.

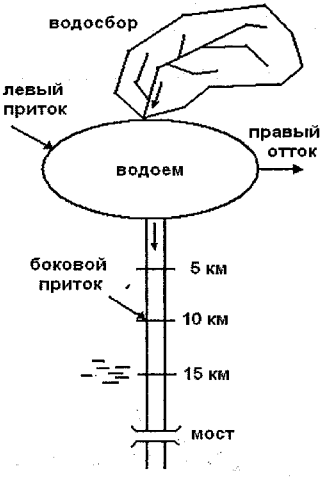


Рис. 1.4. Схема водохозяйственного объекта.

Известно также, что на 10-м километре есть боковой приток, вызывающий изменение уровня в главной реке 0.5 м/ч в точке слияния; на 15-м километре имеет место взаимодействие речных и грунтовых вод, с коэффициентом взаимодействия (или фильтрации) $k_ф = 0.1$ 1/ч.

* В учебном задании использована вымышленная ситуация, не соответствующая по численным значениям расчетных величин какому-либо реальному объекту (это замечание относится также к п. 3.5).

Из непосредственных измерений на гидрологических постах следует, что расходы левого притока и правого оттока не зависят от уровня воды в водоеме и равны $Q_л = 19$ м^3/ч; $Q_п = 21$ м^3/ч.

Кроме того известна следующая дополнительная информация:

- для бассейна верхнего притока – время добегания $T_{доб} = 2$ ч; коэффициент стока $k = 0.16$; $Q|_{t=0} = 10$ м^3/ч;
- для водоема – морфометрический коэффициент $k_{морф} = 0.46$ 1/ч, $H|_{t=0} = 1$ м, площадь $F = 19.2$ м^2, которая не меняется при изменении уровня;
- для руслового стока на нижней реке – скорость распространения волны по руслу $a = \text{const} = 1760$ м/ч, начальное наполнение русла 1 м вдоль всей реки, движение воды в русле управляется (создается) изменением уровня воды в водоеме;
- для насыщенной зоны грунтовых вод – грунтовые воды имеют горизонтальную поверхность по обеим координатам x и y вне зависимости от того, какие процессы происходят в реке; внешнее воздействие на зону насыщения отсутствует, т. е. равно нулю; происходит только взаимодействие с речными водами, причем так, что $H_{гр} = = \text{const} = 1$ м на всем промежутке времени, на котором мы интересуемся процессами в данной системе.

В качестве внешнего воздействия на всю рассматриваемую систему задается метеопрогноз по осадкам на бассейн верхнего притока в водоем (табл. 1.2):

Таблица 1.2

Внешнее воздействие на систему

t ч	0	1	2	3	4	5
X м^3/ч	200	500	500	300	150	0

Необходимо рассчитать процесс формирования волны по всем элементам системы и определить максимальную высоту (амплитуду) волны на 20-м километре в районе моста, а также время прохождения максимума под мостом. Выяснить, произойдет ли авария?

Полезные указания, упрощающие расчеты:

1. При рассмотрении процесса формирования стока на верхнем бассейне не учитывать влияния на этот процесс поднятия уровня в водоеме.

2. При расчете подъема уровня в водоеме не учитывать изъятия части стока нижней рекой.

3. Шаги по времени брать $\Delta t = 1$ ч, а по координате $x = 5000$ м.

4. При переходе к 15-му километру надо использовать условие стыковки модели руслового стока и насыщенной зоны.

1.3.2. Пример решения

1. Модель склонового стока (рис. 1.5).

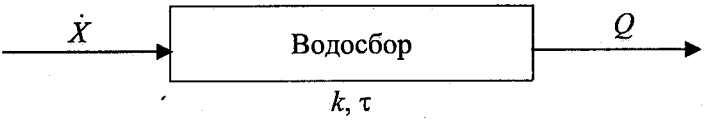


Рис. 1.5. Схема моделируемого объекта.

Расчеты выполняются по формулам:

$$\frac{dQ}{dt} = -\frac{Q}{\tau} + \frac{k\dot{X}}{\tau};$$

$$\frac{Q_{i+1} - Q_i}{\Delta t} = -\frac{Q_i}{\tau} + \frac{k\dot{X}_i}{\tau};$$

$$Q_{i+1} = Q_i - \frac{Q_i}{\tau}\Delta t + \frac{k\dot{X}_i}{\tau}\Delta t;$$

$$Q_1 = Q_0 - 0.5\,Q_0 + 0.08\,\dot{X}_0 = 10 - 5 + 0.08 \cdot 200 = 21.0;$$

$$Q_2 = 21.0 - 21.0/2 + 0.08 \cdot 500 = 50.5;$$

$$Q_3 = 50.5 - 50.5/2 + 0.08 \cdot 500 = 65.2;$$

$$Q_4 = 65.2 - 65.2/2 + 0.08 \cdot 300 = 56.6;$$

$$Q_5 = 56.6 - 56.6/2 + 0.08 \cdot 150 = 40.3;$$

$$Q_6 = 40.3 - 40.3/2 + 0.08 \cdot 0 = 20.2.$$

2. Модель водоема (рис. 1.6).

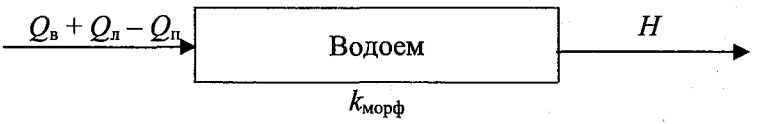


Рис. 1.6. Схема моделируемого объекта ($Q_в$ – приток воды с верхнего водосбора).

Расчеты выполняются по формулам:

$$\frac{dH}{dt} + k_{морф} H = \frac{Q_в + Q_л - Q_п}{F};$$

$$\frac{H_{i+1} - H_i}{\Delta t} + k_{морф} H_i = \frac{Q_{в_i} + Q_л - Q_п}{F};$$

$$H_{i+1} = \frac{Q_{в_i} + Q_л - Q_п}{F} \Delta t - k_{морф} H_i \Delta t + H_i;$$

$H_1 = (10 - 2)/19.2 - 0.46 \cdot 1 + 1 = 0.96;$

$H_2 = (21.0 - 2)/19.2 - 0.46 \cdot 0.96 + 0.96 = 1.51;$

$H_1 = (50.5 - 2)/19.2 - 0.46 \cdot 1.51 + 1.51 = 3.34;$

$H_1 = (65.2 - 2)/19.2 - 0.46 \cdot 3.34 + 3.34 = 5.10;$

$H_1 = (56.6 - 2)/19.2 - 0.46 \cdot 5.10 + 5.10 = 5.60;$

$H_1 = (40.3 - 2)/19.2 - 0.46 \cdot 5.60 + 5.60 = 5.02;$

$H_1 = (20.2 - 2)/19.2 - 0.46 \cdot 5.02 + 5.02 = 3.66.$

3. Модель руслового стока (рис. 1.7).

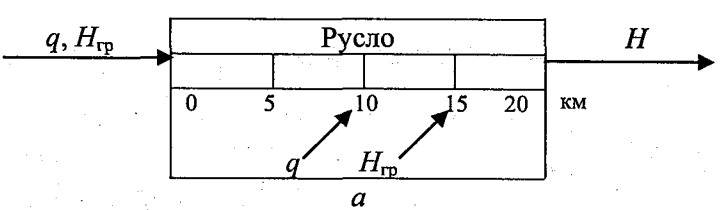


Рис. 1.7. Схема моделируемого объекта

Расчеты выполняются по формулам (результаты в табл. 1.3):

$$\frac{\partial H}{\partial t} + a\frac{\partial H}{\partial x} = q;$$

$$\frac{H_{i+1,j} + H_{i,j}}{\Delta t} + a\frac{H_{i,j} - H_{i,j-1}}{\Delta x} = q_{i,j};$$

$$H_{i+1,j} = H_{i,j} - a\frac{\Delta t}{\Delta x}(H_{i,j} - H_{i,j-1}) + q_{i,j}\Delta t;$$

$$H_{i+1,5} = H_{i,5} - 0.352(H_{i,5} - H_{i,0});$$

$$H_{1,5} = 1 - 0.352\cdot(1 - 1) = 1.00;$$

...

$$H_{8,5} = 4.33 - 0.352\cdot(4.33 - 3.66) = 4.09;$$

$$H_{i+1,10} = H_{i,10} - 0.352(H_{i,10} - H_{i,5}) + 0.5\Delta t;$$

$$H_{1,10} = 1 - 0.352\cdot(1 - 1) + 0.5 = 1.50;$$

...

$$H_{9,10} = 4.62 - 0.352\cdot(4.62 - 4.09) + 0.5 = 4.93;$$

$$H_{i+1,15} = H_{i,15} - 0.352(H_{i,15} - H_{i,10}) + k_{\text{ф}}(H_{\text{гр}} - H_{i,15});$$

$$H_{1,15} = 1 - 0.352\cdot(1 - 1) + 0.1\cdot(1 - 1) = 1.00;$$

...

$$H_{10,15} = 3.26 - 0.352\cdot(3.26 - 4.93) + 0.1\cdot(1 - 3.26) = 3.62;$$

$$H_{i+1,20} = H_{i,20} - 0.352(H_{i,20} - H_{i,15});$$

$$H_{1,20} = 1 - 0.352\cdot(1 - 1) = 1.00;$$

...

$$H_{11,20} = 2.58 - 0.352\cdot(2.58 - 3.62) = 2.95.$$

Таблица 1.3

Уровни воды в русле

x км / *i* час	0	5	10	15	20
11					2.95
10				3.62	2.58
9			4.93	3.26	2.21
8		4.09	4.62	2.79	1.89
7	3.66	4.33	4.01	2.34	1.65

i час \ x км	0	5	10	15	20
6	5.02	3.95	3.27	1.98	1.47
5	5.60	3.05	2.62	1.75	1.32
4	5.10	1.93	2.23	1.58	1.18
3	3.34	1.17	2.03	1.39	1.06
2	1.51	0.99	1.82	1.18	1.00
1	0.96	1.00	1.50	1.00	1.00
0	1.00	1.00	1.00	1.00	1.00

Полученные результаты в виде гистограмм представлены на рис. 1.8.

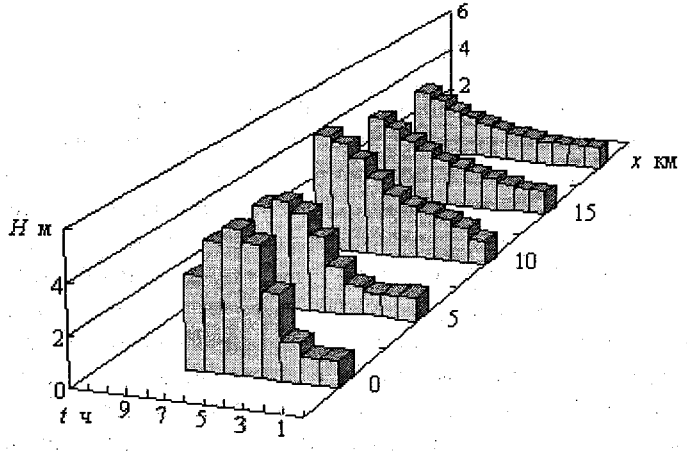


Рис. 1.8. Изменение уровней воды в русле при перемещении кинематической волны.

1.4. Расчетно-измерительные модели

1.4.1. Вывод модели «петли» из системы Сен-Венана

Кроме классического применения моделей для решения различных краевых задач, когда их постоянные или переменные параметры известны заранее, существует область использования моделей в так называемом реальном времени, когда информация о параметрах модели и внешних воздействиях поступает с измерительных приборов непосредственно при решении уравнения. Воз-

можность измерять те или иные характеристики изучаемых гидрологических процессов часто позволяет существенно упростить математическую модель.

Для учета речного стока широко используются модели расхода воды, основанные на связи $Q = f(H)$, например в виде полинома $Q = a_0 + a_1 H + a_2 H^2 + \ldots$ Однако эта связь может быть и неоднозначной. Остановимся на причинах этой неоднозначности, опираясь на систему уравнений Сен-Венана:

$$-\partial H / \partial x = Q^2 / (C^2 R F^2) + [1/(gF)]\partial Q / \partial t + [2\alpha Q / (gF^2)]\partial Q / \partial x, \quad (1.19)$$

$$\partial Q / \partial x + \partial F / \partial t = 0, \quad (1.20)$$

где C – коэффициент Шези; R – гидравлический радиус; g – ускорение свободного падения; α – коэффициент Кориолиса.

Система уравнений Сен-Венана довольно полно описывает в одномерной гидравлической идеализации движение неустановившегося потока в русле. В результате ее решения находят функции $Q = f(x, t)$ и $H = f(x, t)$ при известных начальных и граничных условиях. Однако в гидрометрии имеют дело с конкретными закрепленными створами (т. е. координата x фиксированная, $x = x_0$) и предполагают, что морфометрические характеристики створа B, F, R и др. (здесь B – ширина), а также ход уровня во времени $H = f(t)$ легко измерить. Если предположить, что для фиксированного створа с координатой x_0 из измерений известны зависимости $C = f(H)$ и $I = f(t)$ (здесь $I = -\partial H / \partial x$ – уклон водной поверхности), то подставляя производную $\partial Q / \partial x = -\partial F / \partial t$ из уравнения неразрывности (1.20) в динамическое уравнение (1.19), последнее можно записать в виде дифференциального уравнения в обыкновенных производных:

$$dQ / dt = f_1(x_0, t) Q^2 + f_2(x_0, t) Q + f_3(x_0, t), \quad (1.21)$$

где $f_1(x_0, t) = -g / (C^2 R F)$, $f_2(x_0, t) = (2\alpha / F)\, \partial F / \partial t$, $f_3(x_0, t) = gIF$.

Решить его – значит найти для фиксированного створа зависимость $Q = f(t)$ при известных начальном расходе воды, коэффи-

циентах $f_1(x_0,t)$ и $f_2(x_0,t)$ и свободном члене $f_3(x_0,t)$. Так как $F=f(H)$, $C==f(H)$ и $R=f(H)$, то при измеренных значениях H и I имеем $Q=f(H,\partial H/\partial x,\partial H/\partial t)$, так как $\partial H/\partial x=-I$, а $\partial H/\partial t$ определяется функцией $H(t)$ при непрерывном или дискретном, но достаточно частом измерении H. Таким образом, если при равномерном режиме зависимость $Q=f(H)$ однозначна и определяется формулой Шези, то при плавно изменяющемся движении она определяется нелинейным дифференциальным уравнением первого порядка с переменными коэффициентами, которое, по существу, является уравнением петлеобразной зависимости $Q=f(H)$. (Формально-математически – это так называемое уравнение Риккати.) Можно показать, что в более общем случае неплавноизменяющегося движения имеет место зависимость $Q=f(H,\partial H/\partial x,\partial H/\partial t,\partial^2 H/\partial x^2,\partial^2 H/\partial x\partial t,...)$, т. е. расход воды зависит не только от уровня и уклона, но и от кривизны свободной поверхности, что, однако, является для практической гидрометрии редким случаем. Следует обратить внимание на то, что хотя традиционно в гидрометрии считают, что расход воды определяется уровнем, но на самом деле ситуация обратная.

Полученное уравнение петли можно решить численными методами (например, Рунге–Кутты) или на аналоговой ЭВМ в реальном масштабе времени. В последнем случае имеем устройство для измерения расхода воды, первичными измерительными преобразователями которого являются уровнемер и уклономер. Так как в модель входят непосредственно измеряемые параметры, то естественно ее назвать измерительно-расчетной. Однако при таком подходе возникает сложный вопрос о корректности ввода измерительной информации, которая заведомо не может быть очень точной, в соотношения, получаемые из законов сохранения и в этом смысле являющимися точными, т. е. встает проблема стыковки математической модели с данными измерений.

1.4.2. Исходная гидрометрическая информация и пример расчета

Для расчета по уравнению Риккати (1.21) нужна следующая исходная информация: $H=f(t)$, $I=f(t)$, $F=f(H)$, $R=f(H)$, а также значения коэффициентов и начального условия, которые примем

следующими $n = 0.03$; $a = 10$ (см. последнюю колонку в табл. 1.4); $\alpha = 1$; $g = 9.81$ м^2/с, $Q_0 = 20$ м^3/с. Зависимости (в табличном виде) представлены в табл. 1.4.

Таблица 1.4

Исходная информация

t, мин	H, м	R, м	I	F, м^2 ($F = aH^2$)
0	1.00	1.00	0.0010	10.00
5	2.50	2.70	0.0017	62.50
10	3.00	4.00	0.0020	90.00
15	2.75	3.00	0.0017	75.63
20	2.00	2.00	0.0014	40.00
25	1.25	1.25	0.0012	15.63
30	1.00	1.00	0.0010	10.00

Предварительные расчеты заключаются в последовательном нахождении в уравнении (1.21) численных значений функций f_1, f_2 и f_3 (табл. 1.5, 1.6 и 1.7):

1. Определяем $f_1 = \dfrac{-g}{C^2 RF}$, где $C = R^{1/6}/n$ (табл. 1.5).

Таблица 1.5

Функция $f_1(t)$

t	H	C^2RF	$f_1 \cdot 10^{-4}$
0	1.00	11111	–8.83
5	2.50	261075	–0.38
10	3.00	635040	–0.15
15	2.75	363605	–0.27
20	2.00	111502	–0.88
25	1.25	23382	–4.20
30	1.00	11111	–8.83

2. Определяем $f_2 = \dfrac{2\alpha}{F} \cdot \dfrac{\partial F}{\partial t}$; $\dfrac{\partial F}{\partial t} \approx \dfrac{\Delta F}{\Delta t}$, $\Delta t = 300$ с (табл. 1.6).

Таблица 1.6

Функция $f_2(t)$

t	H	F	$\partial F / \partial t$	f_2
0	1.00	10.0	0	0
5	2.50	62.5	0.175	0.0056
10	3.00	90.0	0.092	0.0020

t	H	F	$\partial F / \partial t$	f_2
15	2.75	75.6	–0.048	–0.0013
20	2.00	40.0	–0.119	–0.0060
25	1.25	15.6	–0.081	–0.0104
30	1.00	10.0	–0.190	–0.0038

3. Определяем $f_3 = gIF$ (табл. 1.7).

Таблица 1.7

Функция $f_3(t)$

t	I	F	f_3
0	0.0010	10.0	0.098
5	0.0017	62.5	1.042
10	0.0020	90.0	1.766
15	0.0017	75.6	1.261
20	0.0014	40.0	0.549
25	0.0012	15.6	0.184
30	0.0010	10.0	0.098

В заключение находим значения расхода воды методом Эйлера ($\Delta t = 60$ с):

$$Q_i = Q_{i-1} + 60S(Q_{i-1}, t_{i-1}),$$

где

$$S_{i-1} = f_1(t_{i-1})Q_{i-1}^2 + f_2(t_{i-1})Q_{i-1} + f_3(t_{i-1}).$$

Так как в таблицах 1.5–1.7 информация представлена с дискретностью 5 мин, а для устойчивости вычислений расчеты надо проводить с шагом в одну минуту, то необходимо выполнять интерполяцию функций f_1, f_2 и f_3.

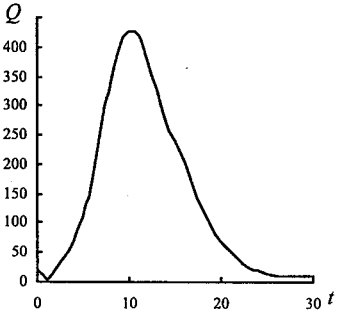


Рис. 1.9. Гидрограф стока.

Приведем вычисления на первых трех шагах:

$$Q_1 = Q_0 + 60(f_1(t_0)Q_0^2 + f_2(t_0)Q_0 + f_3(t_0)) =$$
$$= 20 + 60(-0.000883 \cdot 20^2 + 0 \cdot 20 + 0.098) = 4.69 \text{ м}^3/\text{с};$$

$$Q_2 = Q_1 + 60S(Q_1, t_1) = 4.69 + 60(-0.00055 \cdot 4.69^2 + 0.0016 \cdot 4.69 +$$
$$+ 0.26) = 20.0 \text{ м}^3/\text{с};$$

$$Q_3 = Q_2 + 60S(Q_2, t_2) = 20.0 + 60(-0.00035 \cdot 20.0^2 + 0.0029 \cdot 20.0 +$$
$$+ 0.46) = 42.7 \text{ м}^3/\text{с} \text{ и т. д.}$$

Полученные результаты (расходы воды на расчетном интервале) представлены на рис. 1.9.

2. Элементы языка C++ (*Си* с классами)

2.1. Структура программ на C++. Среда разработки программ и создание исполняемого файла

Основные блоки, из которых состоит программа на языке C++, представлены на рис. 2.1.

Библиотеки – это готовые «куски» программы (они называются классами, о них речь пойдет ниже). Они могут быть **стандартными**, которые выполняют задачи, одинаковые для любых программ (например, ввод и вывод на монитор информации) или пользовательскими (авторскими).

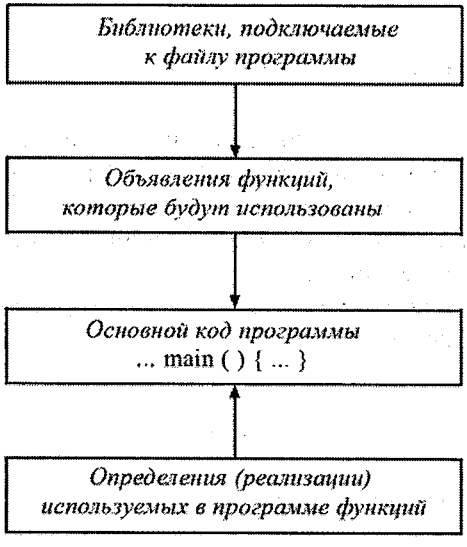


Рис. 2.1. Основные блоки программа на языке C++.

Например,

```
1. # include <iostream>
2. # include "student.h"
3. using namespace std;
```

Читаются эти строчки так:

1. Подключить стандартные классы ввода–вывода информации под именем iostream.Имя берется в угловые скобки (это признак того, что речь идет о стандартных классах).

2. Подключить пользовательский «класс» (скорее это файл, содержащий «много чего») под именем "student.h". Кавычки "..." отличают данный класс от стандартных.

3. Использовать стандартное пространство имен, только из стандартной библиотеки. (Эта строчка с директивой using может находиться в любом блоке программы.)

Объявление функций – это указание имен функцией и некоторых их свойств без описания того, что они конкретно будут делать в программе. (Это все равно что объявить состав футбольной команды «Зенит» без указания кто защитник, нападающий и т. д.).

Пример.

```
int Add (int x, int y);
```

Объявляется функции под названием Add, на «вход» которой надо подавать два параметра x и y типа int (целые числа), а на «выходе» значение функции также должно быть целым числом. Говорят так: функции Add **возвращает** целочисленное значение (куда возвращает, разберем ниже).

Основной код программы – это конкретные указания, что должна делать программа.

Пример:

```
0    int main ()
1    {
2      int v, F, Q;
3      cout << "Enter two numbers: ";
4      cin >> v;
5      cin >> F;
6      Q = Add(v, F);
7      cout << "Result is:\n" << Q;
8      system("pause");
9      return 0;
10   }
```

В **определении** (**реализации**) функции указывается, что конкретно Add должна делать с **аргументами** «v» и «F» (т. е. конкретными значениями параметров x и y). Строка объявления функции заканчивается точкой с запятой, в определении такого нет:

```
int Add (int v, int F)
{
  return  (v*F);
}
```

Теперь давайте запишем программу полностью (назовем ее Hello):

```
0     # include <iostream>
1     using namespace std;
2     int Add (int x, int y);
3     int main ()
4     {
5       int v, F, Q;
6       cout << "Enter two numbers: ";
7       cin >> v;
8       cin >> F;
9       Q = Add(v, F);
10      cout << "Result is:\n" << Q;
11      system("pause");
12      return 0;
13    }
14    int Add (int v, int F)
15    {
16      return (v*F);
17    }
```

В программе строку номер 0 следует читать так: паунд (# – символ фунта) инклуд (include) иострим (iostream). Этой строкой подключается стандартная библиотека ввода–вывода. В первой строке сообщается компилятору, что будет использоваться стандартное пространство имен std. Вторая – объявление функции Add, которая имеет два входных целочисленных параметра и возвращает целочисленное значение. С третьей строки начинается основной код программы, который в конце ограничен фигурной скобкой (13-я строка). Шестая строка содержит объект вывода cout. После выполнения этой строки на экране появится Enter two numbers: . В 7-й и 8-й строках происходит считывание чисел, введенных пользователем, и инициализация переменных v и F этими числами (пере-

менным v и F присваиваются введенные значения). В строке 9 происходит вызов функции Add (14-я строка) и переменная Q инициализируется возвращаемым значением функции. Строка 11-я приостанавливает выполнение программы до нажатия любой клавиши. Двенадцатая строка убирает «мусор» после выполнения программы. С 14-й строки по 17-ю идет реализация функции Add.

Эта запись пока существует только на бумаге. Как ввести ее в машину и что нужно сделать, чтобы ее выполнить? Для этого существует так называемая **среда разработки программ**.

Представим операционную систему (например, Windows), которая управляет работой ПЭВМ, в виде больницы. В больнице много чего: и палаты для больных, и столовая, и бухгалтерия, и ординаторская, и склад и т. д. Написанная нами программа – «больной». Его помещают на стол в операционную, рис. 2.2:

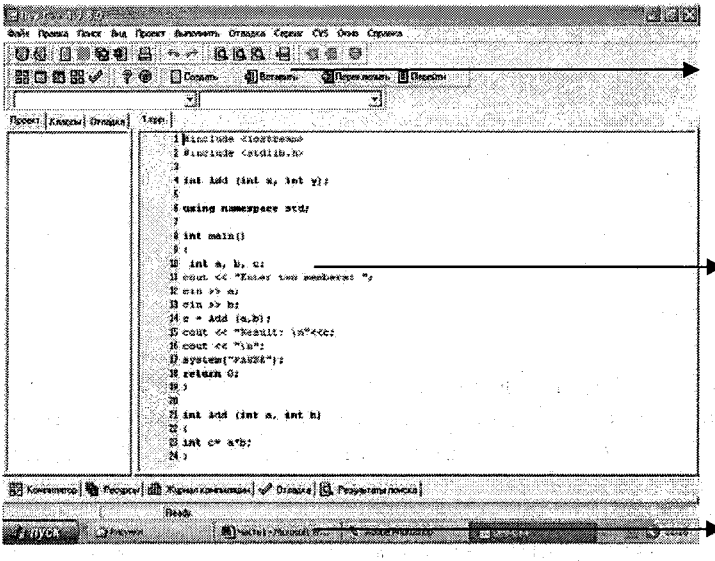


Рис. 2.2. «Больничная» метафора программы.

Эта операционная и называется средой разработки программ (или, не совсем верно, компилятором). «Операционная» управляется диспетчером самой больницы (т. е. операционной системой Windows).

Среда разработки программ включает:

1. Текстовой редактор – для набора и редактирования текста программ.

2. Препроцессор – для подключения библиотек к основному коду программы.

3. Компилятор – для перевода языка C++ в машинный язык Ассемблер.

4. Компоновщик – для компоновки (сборки) блоков программ в единое целое.

Процесс превращения исходного кода программы, набранного в текстовом редакторе, в исполняемый файл можно изобразить так (рис. 2.3):

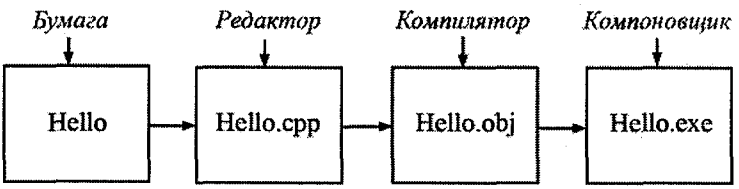


Рис. 2.3. Иллюстрация эволюции исходного кода программы.

В программе реализованы две функции; целесообразно уделить им внимание сейчас, хотя функциям будет посвящен отдельный раздел.

1. **int** main ( )

{ ... }

Функция main присутствует во всех программах и управляется операционной системой. В круглых скобках могут указываться параметры, а если не указаны, то они задаются по умолчанию (как в данном случае).

2. Q = Add (v, F);

Когда программа доходит до этой строки, то управление передается ей (т. е. функции Add). Машина делает то, что требует функция и после присвоения переменной Q значения Add (v, F) управление снова возвращается к основному коду.

Можно предложить следующую аналогию взаимодействия программы (основного кода) и функции, которая присутствует в программе. Рисуем карандашом (это работает программа) и он сло-

мался. Мы вынуждены прервать рисование (т. е. выполнение основного кода программы) и заняться починкой карандаша (т. е. передать управление нашими действиями функции «Починка»). После того как карандаш заострили, управление передается основному коду, т. е. процессу рисования.

Функция может не возвращать в основную программу никакого значения:

```
void Print( )
{
cout << "Hurrah, Hydrology!";
}
```

Если эта функция будет помещена в программу, то на экране просто появится Hurrah, Hydrology!

Вернуть значение из функции в программу можно так:

```
int Add (int v, int F)
{
return (v*F);
}
```

Замечание. Функция может быть объявлена не как у нас, а в так называемом **классе** (рассматривается ниже), и тогда она называется **методом**, который занимается манипуляцией данными класса. Метод – это функция, которая является частью класса. Иногда различие между ними не делают (по крайней мере, словесно).

2.2. Типизация данных

2.2.1. Резервирование памяти и типы переменных

Данные, которые используются в программе, могут быть разных типов. Разнообразие этих типов и является предметом данной главы.

Мы уже имели дело с переменными: int v, F, Q. Но что такое «v» в реальности, а не на бумаге? Переменная «a» – это просто место в оперативной памяти компьютера, где можно хранить конкретное значение переменной «v» и извлекать его.

Переменные хранятся в ячейках, которые можно представить, как показано на рис. 2.4.

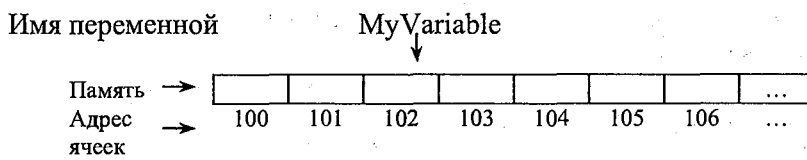


Рис. 2.4. Хранение переменных в ячейках памяти.

Каждая ячейка имеет размер в один байт, что равняется 8 битам. Один бит – это, образно, лампочка: горит – 1, не горит – 0 (рис. 2.5).

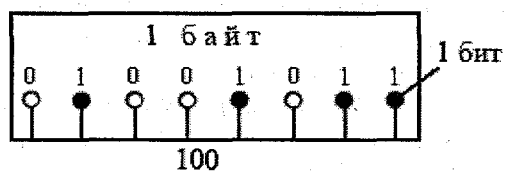


Рис. 2.5. Образное представление бита и байта.

Когда мы указываем тип переменной, то как бы говорим компилятору, сколько ячеек он должен зарезервировать, чтобы поместить в них то или иное значение. Имя переменной – это просто надпись на ячейке, а конкретное значение переменной – это последовательность нулей и единиц: 0 1 1 1 0 0 и т. д.

Основные типы переменных представлены в табл. 2.1.

Таблица 2.1

Типы, размер в байтах и диапазон значений данных

Тип (встроенный)	Размер, байт	Значение
I. Логический тип		
bool	1	true или false
II. Числовой целый тип		
unsigned short int	2	от 0 до 65 535
short int	2	от -32 768 до 32 767
unsigned long int	4	от 0 до 4 294 967 295
long int	4	от -2 147 483 648 до 2 147 483 647
int (16 разрядов)	2	от -32 768 до 32 767
int (32 разряда)	4	от -2 147 483 648 до 2 147 483 647
unsigned int (16 разрядов)	2	от 0 до 65 535

unsigned int (32 разряда)	4	от 0 до 4 294 967 295
III. Символьный тип		
char	1	256 значений символов
IV. Числовой вещественный тип		
float	4	от 1,2e-38 до 3,4e38
double	8	от 2,2e-308 до 1,8e308
V. Отсутствие типа		
void		

Существуют еще невстроенные типы. Например, тип string (строка) «this is …». Эти типы надо подключать через стандартные или пользовательские библиотеки.

Так как системы компьютеров могут иметь разную разрядность (16, 32, 64), то для некоторых типов размеры могут отличаться от приведенных. Конкретный размер можно определить с помощью функции sizeof ( ) следующим образом:

```
0    // Действие функции sizeof()
1    # include <iostream>
2    int main ()
3    {
4    using std::cout;
5    cout<<"Size of a float is:\t"
        << sizeof(float)<<" bytes.\n";
6    cout<<"Size of a double is:\t"
        << sizeof(double)<<" bytes.\n";
7    cout<<"Size of an int is:\t"
        << sizeof(int)<<" bytes.\n";
8    cout<<"Size of a long is:\t"
        << sizeof(long)<<" bytes.\n";
9    cout<<"Size of a short is:\t"
        << sizeof(short)<<" bytes.\n";
10   cout<<"Size of a char is:\t"
        << sizeof(char)<<" bytes.\n";
11   cout<<"Size of a bool is:\t"
        << sizeof(bool)<<" bytes.\n";
12   system("pause");
     return 0;
13   }
```

Результат:

```
Size of a float is:    4 bytes.
Size of a double is:   8 bytes.
Size of an int is:     4 bytes.
Size of a long is:     4 bytes.
Size of a short is:    2 bytes.
Size of a char is:     1 bytes.
Size of a bool is:     1 bytes.
```

В строке 4 сообщается, что из стандартного пространства имен будет использован только объект вывода cout. После выполнения строк 5–11 на экране появятся возвращаемые значения функции sizeof при разных аргументах (float, double, int, long, short, char, bool). В программе встречаются специальные символы (\n, \t), о которых речь пойдет в п. 2.2.3.

2.2.2. Определение (задание) переменных

Объявление переменной производится следующим образом:

```
int main ( )
{
 unsigned short x;
  ..................
 float Area;
  ..................
}
```

Объявляем, что x – это беззнаковая целочисленная переменная, для нее надо зарезервировать две ячейки памяти. Объявляем, что переменная Area будет иметь вещественное значение и для нее нужно выделить 4 ячейки памяти.

Переменные могут обозначаться как угодно, но пытаются придерживаться определенных соглашений об именовании:

1. my_var;
2. myVar // верблюжья нотация
3. iVar // венгерская нотация (int Var ↔ iVar; float Var ↔ fVar)

Для обозначения переменных нельзя применять так называемые ключевые слова C++, представленные в табл. 2.2.

Таблица 2.2

Ключевые слова C++

asm	else	new	this
auto	enum	operator	throw
bool	explicit	private	true
break	export	protected	try
case	extern	public	typedef
catch	false	register	typeid
char	float	reinterpret_cast	typename
class	for	return	union
const	friend	short	unsigned
const_cast	goto	signed	using
continue	if	sizeof	virtual
default	inline	static	void
delete	int	static_cast	volatile
do	long	struct	wchar_t
double	mutable	switch	while
dynamic_cast	namespace	template	

and	bitor	not_eq	xor
and_eq	compl	or	xor_eq
bitand	not	or_eq	

Можно объявить сразу несколько переменных:

long int Area, width, length;
char a, b;

Объявляя переменную, например int myVar; мы просто навешиваем на ячейки с определенными адресами (их знает машина) ярлык myVar. Однако в самих ячейках может находится все что угодно (мусор). Чтобы его убрать, надо не только объявить переменную, но и задать ее конкретное значение или, как говорят, инициализировать:

int myVar = 5;

Теперь в ячейках с ярлыком myVar помещен не мусор, а число 5, которое является значением переменной myVar. (Это еще можно

сравнить, метафорически, с заданием начального условия: есть переменная Q, а есть ее значение при $t = t_0$: $Q|_{t=t_0} = 5$.)

Приведем пример программы.

```
0    //Использование переменных
1    # include <iostream>
2    int main ()
3    {
4    using std::cout;
5    unsigned short int Width = 3, Length = 6;
6    unsigned short int Height;
7    Height = 3;
8    //объявить переменную типа unsigned short и
9    // инициализировать ее результатом умножения
10   // Width, Length и Height
11   unsigned short int Volume =(Width*Length* Height);
12   cout << "Width:  " << Width << "\n";
13   cout << "Length: " << Length << std::endl;
14   cout << "Height: " << Height << std::endl;
15   cout << "Volume: " << Volume << std::endl;
16   system("pause");
17   return 0;
18   }
```

Результат:

```
Width:  3
Length: 6
Height: 3
Volume: 54
```

Как обычно, эта программа начинается с блока подключения библиотек, а именно подключается стандартная библиотека ввода/вывода (строка 1). В строке 4 компилятору сообщаться, что будет использоваться объект cout из стандартного пространства имен (std). Объект cout необходим для вывода на экран символьных строк и значений переменных. В строках 13, 14 и 15 пришлось трижды указывать, что объект endl (переводит курсор на новую строку как специальный символ \n) принадлежит стандартному пространству имен (это можно было бы сделать один раз по аналогии со строкой 4, подобный вариант реализован в следующей программе). В программе несколько раз встречается unsigned short int

для объявления переменных (строки 5, 6, 11). Чтобы не писать много слов, его можно заменить псевдонимом (синонимом), используя ключевое слово **typedef** (определение типа, но это не создание нового типа). Перепишем предыдущую программу с использованием псевдонима.

```
0    //Ключевое слово typedef
1    # include <iostream>
2    typedef unsigned short int USI;
3    //определение псевдонима
4    int main ()
5    {
6    using std::cout;
7    using std::endl;
8    USI Width = 3, Length = 6;
9    USI Height = 3;
10   USI Volume = (Width*Length*Height);
11   cout << "Width:  " << Width << endl;
12   cout << "Length: " << Length << endl;
13   cout << "Height: " << Height << endl;
14   cout << "Volume: " << Volume << endl;
15   system("pause");
16   return 0;
17   }
```

Результат:

```
Width:  3
Length: 6
Height: 3
Volume: 54
```

Результат ничем не отличается от результата предыдущей программы. В строке 2 определен псевдоним с помощью ключевого слова **typedef**. Рекомендуется псевдоним писать заглавными буквами.

Существует такое явление, как переполнение регистра переменной. Для пояснения рассмотрим программу.

```
0    // Переполнение регистра
1    # include <iostream>
2    using namespace std;
3    int main ()
4    {
```

```
5       unsigned short int sNumber = 65534;
6       cout << "small number: " << sNumber << endl;
7       sNumber++; // sNumber = sNumber + 1;
8       cout << "small number: " << sNumber << endl;
9       sNumber++;
10      cout << "small number: " << sNumber << endl;
11      sNumber++;
12      cout << "small number: " << sNumber << endl;
13      system("pause");
14      return 0;
15      }
```

Результат:

```
small number: 65534
small number: 65535
small number: 0
small number: 1
```

В программе в строке 5 объявлена переменная sNumber **типа unsigned short int** (целочисленная короткая беззнаковая). Максимальное значение, которое может иметь переменная такого типа, равно 65 535. Переменная инициализирована значением 65 534. Далее выводится значение переменной на экран. В 7-й строке происходит увеличение значения переменной sNumber на единицу. Запись в этой строке эквивалентна записи sNumber = sNumber+1; В 9-й и 11-й строках проделывается аналогичная операция. Диапазон значений переменной определенного типа (в данном случае **unsigned short int**) можно сравнить с циферблатом: после числа 12 опять идет 1, тринадцати никогда не увидим. Так и в программе: последнее число диапазона 65 535, а затем снова 0.

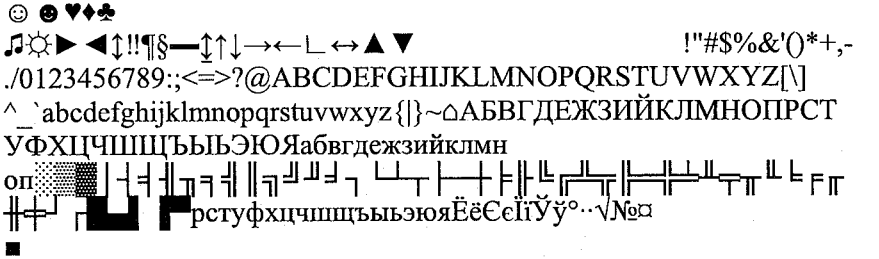
2.2.3. Символьный тип переменной и специальные символы

Символьные переменные (**char**) предназначены для хранения элементов таблицы кодировки ASCII (American Standard Code for Information Interchange – американский стандартный код для обмена информацией).

Можно все символы, закодированные числами в этой таблице, вывести на экран:

```
0    # include <iostream>
1    using namespace std;
2    int main ()
3    {
4    for (int i=0; i < 256; i++)
5      cout << (char)i;
6    system("pause");
7    return 0;
8    }
```

Результат:



Можете набрать, скомпилировать и выполнить эту программку, а понять ее сможете после изучения циклов.

Компилятор C++ распознает некоторые специальные символы (Escape – последовательности), предназначенные для форматирования текста (табл. 2.3).

Таблица 2.3

Специальные символы

Символ	Значение символа
\'	Одинарная кавычка (single quote)
\"	Двойная кавычка (double quote)
\\	Символ обратной косой черты (backslash)
\0	Пустой символ (null)
\000	Восьмеричное число (octal notation)
\a	Звуковой сигнал (bell)

Символ	Значение символа
\b	Забой (backspace)
\f	Перевод страницы (form feed)
\n	Новая строка (new line)
\r	Возврат каретки (carriage return)
\t	Горизонтальная табуляция (tab)
\v	Вертикальная табуляция (vertical tab)
\x	Шестнадцатеричное число (hexadecimal)
\?	Вопросительный знак (question mark)

Косая черта \ («слеш») – указывает, что следующий за ним символ является управляющим.

2.2.4. Константы

В обычной жизни мы легко понимаем, что такое константа, например 5.37.

В языке программирования C++ более широкое ее понимание. В нем существует три вида (типа) констант:

1) литеральные;
2) символьные;
3) перечисляемые.

Литеральные. **int** myAge = 27;

Константа 27 вводится самой программой и ее нельзя изменять (т. е. не myAge, а 27).

Символьные. Это константы, представленные именем. В отличие от простой переменной значение символьной константы также инициализируется, но изменить его уже нельзя.

Почему символьная константа выгоднее литеральной?
Например:

```
student = groups * 25;
// student – число студентов в институте (переменная)
// groups – число групп в институте
// 25 – число студентов в группе (литеральная константа)
```

Но можно использовать символьную константу:

student = groups * s_group;
// s_group – число студентов в группе (символьная константа)
Если число студентов в группе (цифра 25) измениться, то в случае символьной константы не надо будет менять эту цифру в каждой строчке программы, где она встречается, а только один раз изменить значение s_group.

В программе символьная константа вводиться (определяется) следующим образом:

const	**unsigned short int**	s_group	=	25;
ключевое слово	тип символьной константы	имя символьной константы	оператор присвоения	значение символьной константы

В отличие от переменных константы надо инициализировать сразу при объявлении.

Перечисляемые константы. Это как бы создание нового типа данных:

enum	Days	{Sunday, Monday, Tuesday, Wednesday, Thursday, Friday, Saturday};
ключевое слово (перечисление)	название типа	список значений константы

Данное выражение выполняет две задачи: 1) создается перечисляемая константа нового типа с именем Days; 2) определяются символьные константы: Sunday со значением 0, Monday со значением 1 и т. д. Каждому элементу перечисляемой константы соответствует определенное целочисленное значение. По умолчанию первый элемент – 0, а каждый последующий на единицу больше, но можно задать (проинициализировать) и явно:
enum Days {Sunday = 100, Monday, Tuesday = 500, Wednesday, Thursday = 700, Friday, Saturday};
Пример программы.

```
0    // Перечисляемый тип
1    # include <iostream>
2    int main ()
```

```
3     {
4     enum Months {January, February, March, April, May,
5     June, July, August, September, October, November,
6     December};
7     Months today;
8     today = September;
9     if (today == July || today == August)
10       std::cout << "\nVocation!\n";
11    else
12       std::cout << "\nWorking days.\n";
13    system("pause");
14    return 0;
15    }
```

Результат:
Working days.

В программе в строке 4 создается перечисляемая константа нового типа Months. В строке 7 объявляется переменная типа Months и в строке 8 инициализируется значением September, что соответствует целочисленному значению 8. С 9-й строки по 12-ю идет ветвление if (подробно будет рассмотрено позже), в котором проверяется, равняется ли значение переменной today July или August. В случае истинного утверждения выводится сообщение "Vocation!", в противном случае – "Working days.".

Из изложенного следует вывод, что язык C++ типизированный, хотя и терпимый к отклонениям:

```
int myVar = 5.2;
```

Компилятор пропустит такое объявление, но округлит 5.2 до целого числа 5. Так как память сейчас стала дешевой, то на ней можно не экономить, т. е. резервировать с запасом. Но при этом будет теряться быстродействие.

Зачем вообще нужна типизация (есть языки не типизированные)? Типизация нужна для того, чтобы при компиляции проверялось соответствие типов и не возникало ошибок при выполнении программ.

2.3. Операторы

2.3.1. Объединение операторов присвоения и математических операторов

В предыдущем разделе были рассмотрены данные и их типы. Но данные не нужны сами по себе. Они нужны для манипуляции ими, передачи и т. д., т. е. для **действий**.

В языке C++ существует несколько категорий операторов, в частности оператор **присвоения** и **математические** операторы.

Оператор присвоения =. Например:

```
x = a + b;
```

Читается так: присвоить переменной x результат суммирования значений переменных a и b. (Нельзя говорить: x равно a плюс b; оператор равно (= =) появится ниже.)

Мы не будем вдаваться в тонкости определений, но на некоторых нюансах остановимся. Всю строчку

```
x = a + b;
```

называют **выражением** (это тоже оператор). Выражение всегда заканчивается точкой с запятой ;. Если просто ;, то это называется **пустым** оператором.

Еще говорят так: переменной x **возвращается** результат суммирования.

Оператор может быть **составным**:

```
{
temp = x;
x = y;
y = temp;
}
```

это называется **блок кода**.

Существует пять математических операторов:

+	сложения,
–	вычитания,
*	умножения,
/	целочисленное деление,
%	деление по модулю.

Деление: 19/4 = 4 (в остатке 3). Этот остаток можно получить так: 19%4 = 3.

Часто математические операторы **объединяют** с операторами присвоения.

Во многих языках (в Паскале, в частности, и в C++ в том числе) возможна такая запись:

myWeight = myWeight + 2;

Читается так: добавить два к значению переменной myWeight и присвоить результат переменной myWeight.

Однако в C++ существует более простой вариант этой записи:

```
myWeight += 2;
myWeight -= 2;
myWeight %= 2;
myWeight /= 2;
myWeight *= 2;
```

Имеет место приоритет выполнения операторов. Он примерно такой же как в алгебре, например: x = 5 + 3 * 8 = 29. Но можно расставить скобки и изменить приоритет: x = (5 + 3) * 8 = 64.

2.3.2. Инкремент и декремент

Оператор **инкремента** обозначается так:

```
counter ++;
```

Читается следующем образом: увеличить значение переменной counter на единицу. Эта запись эквивалентна следующим:

counter = counter + 1;

counter += 1;

Аналогично обстоит дело и с **декрементом**: --.

Нюансы и трудности в восприятии начинаются из-за того, что оба эти оператора существуют в двух вариантах:

префиксном	и	постфиксном
⇩		⇩
++ myAge		myAge ++

Рассмотрим их. Выражение

int a = ++ x;

сообщает компилятору, что переменную x надо сначала увеличить на единицу (если она 4, то сделать 5), а затем присвоить 5 переменной a (следовательно, будет a = x = 5).

Если имеем выражение

int b = x ++;

(пусть x = 7), то компилятор сначала присвоит b текущее значение x (т. е. 7), затем увеличит x до 8 и оставит ее (т. е. x = 8) «в покое» до следующего вызова. При вызове x будет восьмеркой.

```
0    // Использование инкремента и декремента
1    // (префиксного и постфиксного)
2    # include <iostream>
3    int main()
4    {
5    using namespace std;
6    // инициализировать две целочисленные переменные
7    int x = 100;
8    int y = 100;
9    cout << "x:\t " << x << "\n";
10   cout << "y:\t " << y << "\n";
11   x ++;   //постфиксный инкремент
12   ++ y;   //префиксный инкремент
13   cout << ".....\n";
14   cout << "x:\t " << x << "\n";
15   cout << "y:\t " << y << "\n";
16   cout << ".....\n";
17   cout << "x:\t " << ++ x << "\n";
18   cout << "y:\t " << y -- << "\n";
19   cout << ".....\n";
20   cout << "x:\t " << x << "\n";
21   cout << "y:\t " << y << "\n";
22   system("pause");
23   return 0;
24   }
```

Результат:

```
x:      100
y:      100
.....
x:      101
y:      101
```

```
.............
x:    102
y:    101
.............
x:    102
y:    100
```

В строках 7, 8 объявляются и инициализируются одинаковыми значениями две целочисленные переменные. В строках 9, 10 выводятся их значения. Затем происходит увеличение переменных на единицу с помощью оператора постфиксного и префиксного инкремента. Результаты увеличения, выведенные в строках 14 и 15, одинаковые. В строке 17 опять увеличивается значение переменной x, в данном случае сначала происходит увеличение на единицу, а затем вывод на экран значения переменной. Это подтверждается строками 17 и 20, которые выведут идентичные значения. Уменьшение второй переменной (строка 18) происходит с помощью оператора постфиксного декремента: сначала выводится на экран, а затем уменьшается на единицу значение. Разница между постфиксным и префиксным инкрементом очевидна при использовании этого оператора в совокупности с другими, например операторами вывода, присвоения.

2.3.3. Оператор ветвления программы

Оператор ветвления относится к группе логических операторов, о которых мы будем говорить в следующем пункте. Мы выделили этот оператор, чтобы обыграть понятие **ветвление** программы. Подобные программы действуют не так, как было до сих пор (было: последовательное, **линейное** выполнение программы), а **нелинейным** образом: перескакиванием через несколько строк в зависимости от условий, см. рис. 2.6:

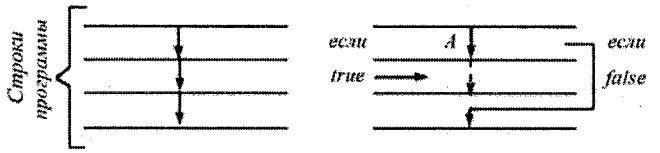


Рис. 2.6. Действие оператора ветвления.

Этот перескок определяется значением логической переменной типа bool. У нее два значения true и false.

В строке *A* что-то сравнивается с чем-то, и в зависимости от результата сравнения происходит переход на ту или иную траекторию, т. е. строка *A* – это своеобразная точка бифуркации (рис. 2.7).

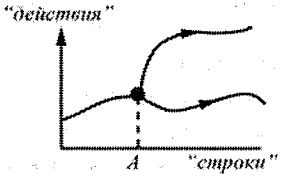


Рис. 2.7. Бифуркационная метафора оператора ветвления.

В точке *A* выполняется операция сравнения. Для сравнения (чего-то с чем-то, например, чисел) используют **операторы отношения**. Их всего шесть (см. табл. 2.4):

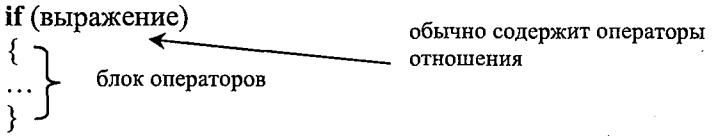
Таблица 2.4

Операторы отношения

Имя	Оператор	Пример	Значение
Равно (equal to)	= =	7 = = 4; 4 = = 4;	false true
Не равно (not equal to)	! =	7 ! = 4; 4 ! = 4;	true false
Больше (greater than)	>	7 > 4; 4 > 4;	true false
Больше или равно (greater than or equal to)	> =	7 > = 4; 4 > = 4;	true true
Меньше (less than)	<	7 < 4; 4 < 4;	false false
Меньше или равно (less than or equal to)	< =	7 < = 4; 4 < = 4;	false true

Операторы отношения играют техническую роль, а ключевое место в процессе ветвления играет **оператор if** (оператор условного перехода или ветвления).

Его синтаксис такой:



Например:

```
if (x > y)
  {
    x = y;
    cout << "x: " << x;
    cout << "y: " << y << endl;
  }
```

(если условие не выполнено, то переход сюда).

Рассмотрим программу ветвления на основе оператора отношения:

```
0   // Использование оператора if
1   // с операторами отношения
2   # include <iostream>
3   using namespace std;
4   int main()
5   {
6   int Width_Rec, Height_Rec;
7   cout << "Enter a width of rectangle: ";
8   cin >> Width_Rec;
9   cout << "Enter a height of rectangle: ";
10  cin >> Height_Rec;
11  cout << endl;
12  if (Width_Rec != Height_Rec)
13      cout << "It's a rectangle!\n";
14  if (Width_Rec < Height_Rec)
15      {
16      cout << "It's a high rectangle.\n";
17      }
18  if (Width_Rec > Height_Rec)
19      {
```

```
20      cout << "It's a wide rectangle.\n";
21      }
22    if (Width_Rec == Height_Rec)
23      cout << "It's a square!\n";
24    system("pause");
25    return 0;
26    }
```

Результат:

```
Enter a width of rectangle: 10
Enter a height of rectangle: 12

It's a rectangle!
It's a high rectangle.
```

В программе пользователю предлагается ввести ширину и высоту прямоугольника. Введенные пользователем числа присваиваются целочисленным переменным Width_Rec и Height_Rec. В строке 12 с помощью оператора if начинается сравнение значений этих переменных. Если значение переменной Width_Rec не равно значению переменной Height_Rec (оператор отношения возвращает true), то будет выполняться строка 13. Затем происходит проверка переменных в строке 14: если значение переменной Width_Rec меньше значения переменной Height_Rec, то выполниться строка 16. Напомним, что если условный оператор в своем теле содержит только одно выражение, то в операторные скобки ({...}) это выражение можно не заключать (строки 13, 15–17). В строке 18 опять происходит сравнение значений переменных. И, наконец, если значения переменных равны, то выполняется строка 23.

Ключевое слово else

Способ ветвления, который был описан, «полулинейный», так как он подразумевал последовательное использование нескольких операторов if (т.е. это не ветвление, а пропуск ненужных строк) для проверки ряда условий. Он работает (отлично, причем), но громоздок (как сито). Сократить код помогает ключевое слово (**директива**) else:

```
if (выражение)
  оператор 1;
else
  оператор 2;
```

Если в выражении получатся true, то выполняется оператор 1, если false, то – оператор 2.
Рассмотрим пример.

```
0     // Использование оператора if
1     // с директивой else
2     # include <iostream>
3     int main()
4     {
5     using std::cout;
6     using std::cin;
7     int First, Second;
8     cout << "Please enter a number-divided: ";
9     cin >> First;
10    cout << "\nPlease enter a number-divider: ";
11    cin >> Second;
12    if ((First % Second) == 0)
13       cout << "\nThanks!\n";
14    else
15       cout << "\nError!\n";
16    system("pause");
17    return 0;
18    }
```

Результат:

```
Please enter a number-divided: 36

Please enter a number-divider: 6

Thanks!
```

В строках 9 и 11 вводятся делимое и делитель. Условие оператора if проверяется в строке 12. Если оно истинно (первое число делиться на второе без остатка), то выполняется строка 13. Если оно ложно, то выполняется строка 15 (строка 13 пропускается).

Таким образом, условный оператор if имеет две формы синтаксиса:

1. **if** (выражение)
   оператор;
   следующий оператор;
2. **if** (выражение)
   оператор 1;
   else
   оператор 2;

Существует еще более сложная конструкция (ветвление в ветвлении) if с вложенным оператором if (if … else в if … else):

```
if (выражение 1)
{
  if (выражение 2)
     оператор 1;
  else
     {
       if (выражение 3)
          оператор 2;
       else
          оператор 3;
     }
}
else
  оператор 4;
```

Оператор 1 выполняется, если истинны выражения 1 и 2. Оператор 2 выполняется, если истинны выражения 1 и 3, а выражение 2 ложно. Оператор 3 выполниться, если выражение 1 истинно, а выражения 2 и 3 ложны. Оператор 4 выполниться при ложном выражении 1.

Пример.

```
0     // Сложный оператор if
1     // с вложенными операторами
2     # include <iostream>
3     using namespace std;
4     int main()
5     {
6     int x, y;
7     cout << "Enter x :\n";
8     cin >> x;
9     cout << "Enter y :\n";
10    cin >> y;
11    cout << "\n";
12    if (x != y)
13    {
14      if (x > 0)
15        {
```

```
16          if (y > 0)
17            cout << "First  square!\n";
18          else
19            cout << "Fourth square!\n";
20        }
21      else
22        {
23         if (y < 0)
24            cout << "Third  square!\n";
25         else
26           cout << "Second square!\n";
27        }
28    }
29    else
30      cout << "We are on an axis!\n";
31    system("pause");
32    return 0;
33    }
```

Результат:

```
Enter x :
3
Enter y :
-5

Fourth square!
```

В программе у пользователя запрашиваются две координаты (x и y) с целью определения четверти тригонометрического круга. Условный оператор начинается в строке 12. Если оператор сравнения возвращает true, то переходим к строке 14. Снова проверяем: если правда, то идем на строку 16 (третий вложенный if). И снова проверяем: в случае истинности выражения, переходим на строку 17. Если в строке 16 – ложь, то идем на строку 19 (оператор else). На строку 23 мы попадаем, когда условный оператор в строке 14 имеет значение false, т. е. в этом случае пропускаются строки 15–20. И, наконец, внешний (первый) условный оператор в строке 12: если x равно y, то, пропуская строки с 13 по 28, мы попадаем на строку 30 (выполняется оператор else первого if).

2.3.4. Логические операторы

Пусть формирование стока моделируется уравнением

$$\frac{dQ}{dt} = -\frac{1}{k\tau}Q + \frac{\dot{X}}{\tau}.$$

Рассмотрим ситуацию, представленную на рис. 2.8, *а*.

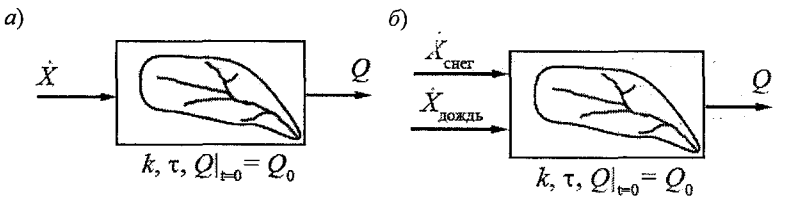


Рис. 2.8. К иллюстрации логических операторов.

Чтобы на выходе бассейна появился расход, надо выполнение (одновременно) 2-х условий: $\dot{X} \neq 0$ и $\tau \neq \infty$.

Изменим ситуацию (рис. 2.8, *б*), условия меняются: $\dot{X}_{снег} \neq 0$ или $\dot{X}_{дожд} \neq 0$.

Это примеры действия логических условий. Они реализуются с помощью логических операторов, табл. 2.5.

Таблица 2.5

Логические операторы

Оператор	Обозначение	Смысл
AND (И)	&&	логическое И (logical AND)
OR (ИЛИ)	||	логическое ИЛИ (logical OR)
NOT (НЕ)	!	унарное НЕ (unary NOT)

Синтаксис логических операторов такой:
AND: **if** ((x > = 7) && (y > = 7))

(Это **логическое выражение** возвратит значение true, если обе переменные (x и y) больше или равны 7, и значение false, если хотя бы одна переменная не удовлетворяет этому требованию.)

OR: **if** ((x > = 7) || (y > = 7)

NOT: **if** (!(x > = 7))

Логические операторы имеют **приоритет**:

1. !
2. AND
3. OR.

Скобки, как обычно, меняют порядок и улучшают понимание. Пример:

if ((x > 7) && y > 7 || z > 7)

if ((x > 7) && (y > 7 || z > 7))

Несколько слов об обозначениях (хорошем тоне). В C++ значения false соответствуют нулю, а true – всем другим числовым значениям, включая отрицательные. Поэтому возможна такая запись:

```
if (x)  // Если x не нуль, присвоить ему нулевое значение
x = 0;
```

Но лучше так:

```
if (x != 0)
x = 0;
```

Аналогично:

```
if (!x)   // истинно, если x равен нулю
if (x = = 0)  // лучше.
```

Существует еще один условный оператор, который работает сразу с **тремя операндами**.

Синтаксис:

(выражение 1) ? (выражение 2) : (выражение 3)

При истинности выражения 1 возвращается значение выражения 2, в противном случае – значение выражения 3.

Пример применения условных операторов.

```
0    // Условный оператор ? :
1    # include <iostream>
2    int main()
3    {
4    int x = 5, y = 7, z;
```

```
5       z = (x> y) ? x : y;
6       std::cout << "z: " << z << "\n";
7       system("pause");
8       return 0;
9       }
```

Результат:
z: 7

Уловный оператор ?: короче эквивалентной конструкции if … else, требующей несколько строк. Строку 5 следует читать следующим образом: «Если x больше y, возвращается значение x, иначе возвращается значение y».

2.4. Глобальные функции

2.4.1. Общие сведения

Функция – это подпрограмма, оперирующая данными и возвращающая значения (рис. 2.9):

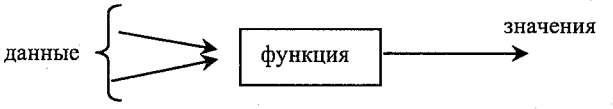


Рис. 2.9. К понятию «функция».

Существуют функции, которые уже **встроены** в часть пакета компилятора, а также – **пользовательские** (разрабатываются самим программистом).

Еще одно важное деление: они могут быть **глобальные** (можно вызывать из любого места программы) и **локальные** (методы объекта класса – о них см. ниже). Рассмотрим глобальные функции.

Выше появилось важное понятие: **вызов** функции (см. п. 2.1). Это понятие хорошо иллюстрирует рис. 2.10.

Функции передаются данные, а она возвращает в программу значения. При объявлении функции это фиксируется так:

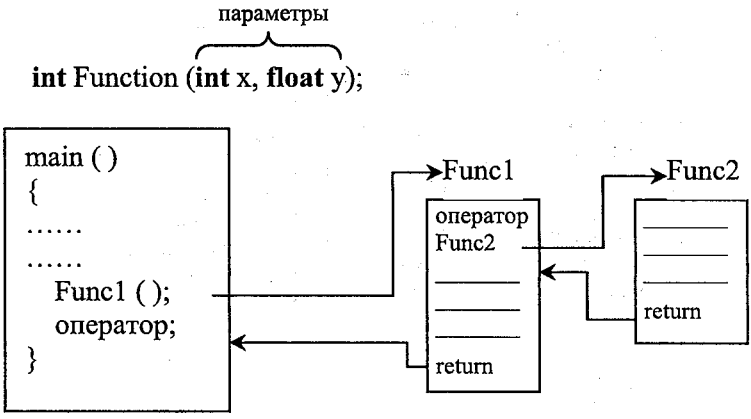
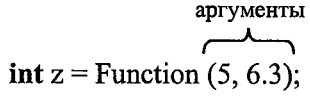


Рис. 2.10. Иллюстрация «вызова функции».

Когда в программе встретится такая функция, то ей надо передать одно значение типа int (целое) и другое – типа float (вещественное). Функция будет что-то делать с ними и вернет в программу значение типа int.

Реально строка в программе (коде) может выглядеть так:



Объявляется переменная целого типа и ей присваивается значение, которое выдаст (говорят **возвратит**) функция, на вход которой поступят два числа: целое 5 и вещественное 6.3 (они называются **аргументами**, т. е. это актуализированные параметры).

2.4.2. Объявления и определения функций

В п. 2.1 уже упоминалось о структуре программ, однако могут быть и другие варианты (рис. 2.11).

Прототипы функций – это **объявление** функций вместе с параметрами, т. е. это выражения, состоящие из типа возвращаемого значения и так называемой **сигнатуры** (рис. 2.12).

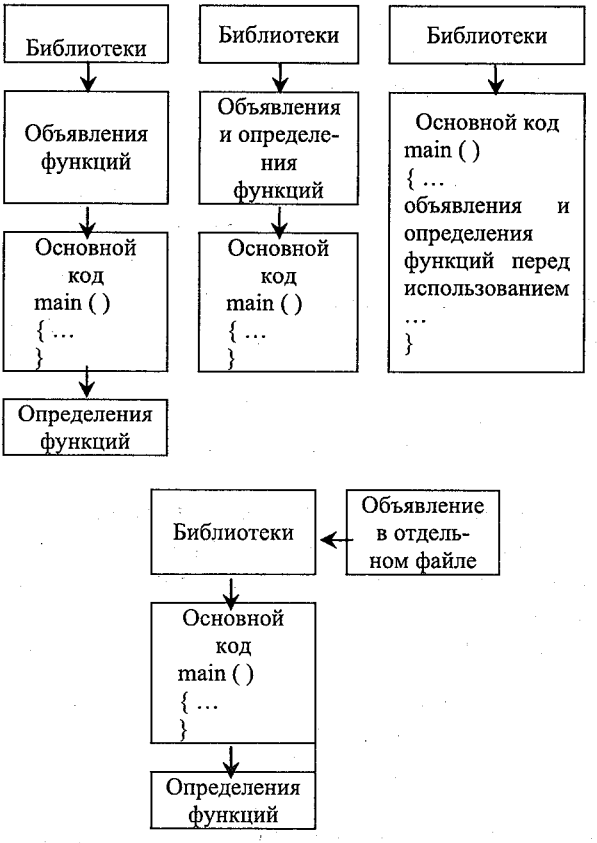


Рис. 2.11. Возможные варианты структур программ.

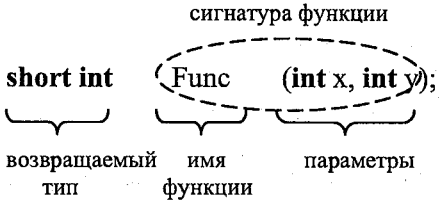


Рис. 2.12. К понятию «сигнатура».

При объявлении функций надо учитывать ряд нюансов:

1. Имена параметров в объявлении не обязательны:

long Func (**int**, **int**);

2. По умолчанию тип возвращаемого значения **int**, т. е. если написать в объявлении Func (**int**, **int**) (забудем указать, «промолчим»), то функция вернет значение **int**.

3. Может быть ситуация, когда функция в основную программу ничего не возвращает:

void Func ( );

т. е., например, что-то выводит на экран.

Определение функции:

```
int Func (int x, int y)
// повторяет объявление без ;
{
...
операторы            // тело функции
...
return (x * y);
}
```

Пример программы с объявлением, определением и реализацией функции.

```
1     // Прототип функции
2     # include <iostream>
3     using namespace std;
4     float Hypotenuse(float a, float b);
5     // прототип функции
6     int main ()
7     {
8     float a, b, c;
9     cout << "First leg: " ;
10    cin >> a;
11    cout << "Second leg: ";
12    cin >> b;
13    c = Hypotenuse(a, b);
14    cout << "Hypotenuse is " << c << endl;
15    system("pause");
16    return 0;
17    }
18    float Hypotenuse (float a, float b)
```

```
19    {
20     return sqrt(a*a + b*b);
21    }
```

Результат:

```
First leg: 3
Second leg: 4
Hypotenuse is 5
```

Прототип функции Hypotenuse объявлен в строке 4. Этой строкой сообщается, что функция принимает два параметра вещественного типа и возвращает (вызывающей функции) значение типа float (дробное число). В строке 6 начинается главная функция программы, которая принимает аргументы по умолчанию и возвращает (операционной системе) целочисленное значение. В строке 8 объявляются вещественные переменные, а в строках 10 и 12 две переменные инициализируются. В строке 13 переменной с присваивается результат выполнения функции Hypotenuse с аргументами a и b. Другими словами, происходит вызов функции Hypotenuse. После этой строки начинает выполняться тело функции (строка 18). Тело функции всегда заключается в фигурные скобки, даже если состоит только из одной строки. В строке 18 – определение функции, которое должно совпадать с объявлением (только без символа ;), но допускается использование разных имен входных параметров (можно при объявлении функции вообще их не указывать). В теле функции, в строке 20, использована стандартная функция извлечения корня sqrt.

2.4.3. Область видимости переменной

Локальные и глобальные переменные. Посмотрим на структуру программ, написанных на языке C++ (рис. 2.13). Этот рисунок поясняет области видимости переменных.

В C++ глобальные переменные, как правило, пытаются не использовать (общедоступность приводит к непредсказуемым последствиям). Поэтому существует их альтернатива – статические переменные-члены (о них – в следующих разделах).

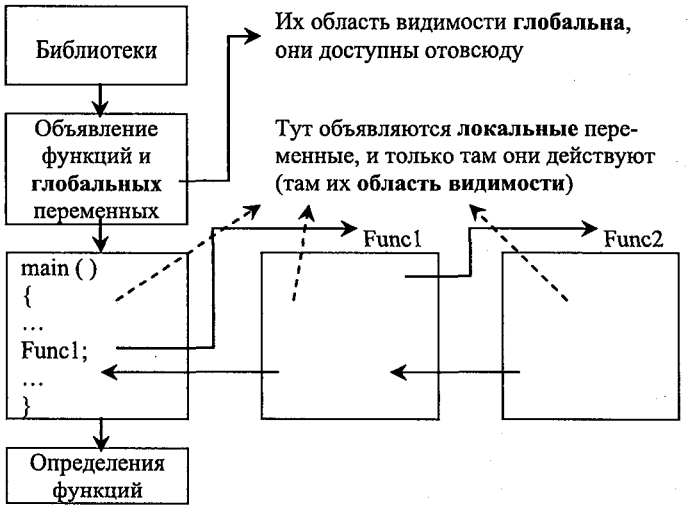


Рис. 2.13. К пояснению локальных и глобальных переменных.

Связь локальных переменных с параметрами функций. Оказывается, что локальными переменными являются не только те, которые объявлены внутри функций, но и переданные им в качестве параметров. Любые манипуляции с параметрами внутри функций не меняют их внешних значений. Результаты манипуляций с переданными параметрами исчезают с прекращением работы функций (остается только значение функции, которое передается в main ( ) («внешний мир»)). По-существу, функции передаются не значения параметров, а их **копии**. Рассмотрим пример.

```
0    // Объявления локальных переменных
1    # include <iostream>
2    using namespace std;
3    float Func(float);
4    int main()
5    {
6    float x, y;
7    cout << "Enter x: ";
8    cin >> x;
9    y = Func(x);
10   cout << "y: " << y << endl;
11   system("pause");
```

```
12      return 0;
13      }
14      float Func (float x)
15      {
16       float y;
17       y = 1.8 * x + 32;
18       return y;
19      }
```

Результат:
Enter x: 36.6
y: 97.88

В программе в строке 3 объявляется функция, которая возвращает и принимает значение вещественного типа. В главной функции объявлены две переменные вещественного типа в строке 6 с именами x и y. В строке 8 переменной x присваивается значение, введенное пользователем с клавиатуры. В строке 9 происходит вызов функции, создается копия переменной x, которая передается в функцию в качестве аргумента (результат работы функции присваивается переменной y). В реализации функции (строки 14–19) объявляется вещественная переменная с таким же именем, что и в строке 6 y. В строке 17 этой переменной присваивается результат вычисления выражения. Следующей строкой следует вернуть из функции значение, так как, несмотря на одинаковые имена, переменные имеют разную область видимости. Работа с переменной y в функции Func никак не сказывается на переменной с таким же именем, объявленной в главной функции.

Локальные переменные внутренних блоков кода (область видимости локальных переменных). Рассмотрим более «тонкие» вещи. Функция может состоять из блоков, ограниченных фигурными скобками. Если переменная определена в каком-то блоке, то только там она и видна (причем только ниже своего определения). Рассмотрим пример.

```
0       // Области видимости
1       // переменных, объявленных внутри блока
2       #include <iostream>
3       using namespace std;
```

```
4       void Func();
5       int main ()
6       {
7        int x = 3;
8        cout << "8:  " << x << endl;
9        Func();
10       {
11       int x = 5;
12       cout << "12: " << x << endl;
13       }
14       cout << "14: " << x << endl;
15       system("pause");
16       return 0;
17      }
18      void Func()
19      {
20       int x = 7;
21       cout << "21: " << x << endl;
22       {
23        cout << "23: " << x << endl;
24        int x = 9;
25        cout << "25: " << x << endl;
26       }
27       cout << "27: " << x << endl;
28      }
```

Результат:

```
8: 3
21: 7
23: 7
25: 9
27: 7
12: 5
14: 3
```

В программе в строке 7 объявляется и инициализируется переменная x и в строке 8 выводится ее значение. В строке 9 происходит вызов функции. В теле функции в строке 20 опять объявляется и инициализируется переменная x, но эта переменная имеет область видимости, ограниченную только функцией Func, т.е. строками 19 и 28. В строке 24 снова объявляется и инициализируется переменная x, но делается это внутри составного оператора, т. е. это локальная переменная с областью видимости от места объ-

явления до закрывающей фигурной скобки (строка 26). Аналогичная ситуация в строках 10–13. В строке 14 главной функции нам видна первая переменная x со значением 3, локально объявленная в блоке в строке 7 она не дает себя «закрыть» другими переменными.

Создание локальных копий при передаче параметров в функцию по значению. При передаче аргументов (т. е. численных значений параметров) в функцию в последней создаются локальные копии аргументов, т. е. значений. Говорят, что параметры передаются по значению. После окончания работы функции в главной функции main ( ) эти значения остаются неизменными.

Пример.

```
0    // Передача параметров по значению
1    # include <iostream>
2    using namespace std;
3    void exchange (int, int, int); // прототип функции
4    int main()
5    {
6     int x = 5,  y = 13, z = 27;
7     cout << "7str  x: " << x << "  y: " << y
     << endl;
8     exchange (x, y, z); // вызов функции
9     cout << "9str  x: " << x << "  y: " << y
     << "  z: " << z << endl;
10    system("pause");
11    return 0;
12    }
13    void exchange (int x, int y, int z)
14    {
15    cout << "15str  x: " << x << "  y: " << y
     << endl;
16    z = x;
17    x = y;
18    y = z;
19    cout << "19str  x: " << x << " y: " << y
     << "  z: " << z << endl;
20    }
```

Результат:
```
7str  x: 5  y: 13
15str  x: 5  y: 13
19str  x: 13 y: 5   z: 5
9str  x: 5  y: 13  z: 27
```

В программе в строке 6 объявлены и инициализированы три целочисленные переменные. В строке 8 происходит вызов функции exchange, которая в ходе своей реализации (строки 14–20) меняет значения переменных x, y и z, переданных функции в качестве аргументов. В строке 19 выводятся на экран новые значения переменных. Как видно из результата (строка 3) обмен значениями состоялся. Но в строке 4 результата, которая соответствует выполнению строки 9 исполняемого кода, значения такие же, как были до вызова функции. Это наглядный пример того, что в функции создаются копии передаваемых аргументов, которые уничтожаются после работы функции.

Глобальные переменные (локальная переменная закрывает глобальную, если их имена совпадают). Повторим, что переменные, определенные вне тела функции (включая main ( )) имеют **глобальную область видимости**. Они доступны из любой функции.

Локальная переменная может иметь имя, совпадающее с именем глобальной переменной, но значение последней измениться не может при изменении значения локальной (т. е. локальная и глобальные переменные остаются разными «сущностями», как Земля и Луна).

Однако локальная переменная может затмить (скрывать) глобальную в том смысле, что внутри функции приоритет выше у локальной переменной (если она объявлена этой функцией).

Рассмотрим пример.

```
0     // Приоритет переменных
1     # include <iostream>
2     using namespace std;
3     void Func ();  // прототип функции
4     int  x = 7, y = 33;//  глобальные переменные
5     int main()
6     {
7     cout << "7str  x: " << x << "  y: " << y << endl;
8     Func ();
```

```
9       cout << "9str  x: " << x << "  y: " << y << endl;
10      system("pause");
11      return 0;
12      }
13      void Func ()
14      {
15       int y = 13; // локальная переменная
16       cout << "16str x: " << x << "  y: " << y << endl;
17      }
```

Результат:
7str x: 7 y: 33
16str x: 7 y: 13
9str x: 7 y: 33

В программе в строке 4 объявлены и определены глобальные переменные. В строке 15 в реализации функции объявлена локальная переменная, имя которой совпадает с именем одной из глобальных переменных. После вызова функции в строке 8 выводятся на экран значения глобальной переменной x и локальной переменной y, т. е. происходит сокрытие глобальной переменной y. После окончания работы функции локальная переменная уничтожается, и в строке 9 выводится значение глобальной переменной y.

2.4.4. Об аргументах функций и возвращаемых значениях

Отметим некоторые нюансы, связанные с аргументами. Они могут быть разных типов: константы, математические и логические выражения, другие функции, возвращающие значения.

Пусть, например, существуют функции: Func_1 ( ), Func_2 ( ) и Func_3 ( ), возвращающие некоторые значения. Допустима следующая запись:

$$\text{result} = \underset{(((x+1)^2+1)^3+1)^4}{\text{(Func\_1}} \; \underset{((x+1)^2+1)^3}{\text{(Func\_2}} \; \underset{(x+1)^2}{\text{(Func\_3}} \; \underset{x}{\text{(Value))));}}$$

Однако в этой абракадабре легко запутаться. Поэтому лучше ввести промежуточные переменные:

```
int Value = 0;
int f1 = Func_3 (Value);  // (0+1)^2 = 1
```

int f2 = Func_2 (f1); // $(1+1)^3 = 8$
int result = Func_1 (f2); // $(8+1)^4 = 6561$

Так что компактный код не значит лучший.

Функции возвращают либо некоторое значение, либо значение типа void, т. е. ничего. Возвращение организуется с помощью ключевого слова **return**, например:
return (Func ( ));

Пример.

```
0    // Несколько операторов return в теле функции
1    #include <iostream>
2    using namespace std;
3    int Func (int, int);
4    int main()
5    {
6    int result = 0;
7    int x, y;
8    cout << "Enter x: ";
9    cin >> x;
10   cout << "Enter y: ";
11   cin >> y;
12   result = Func (x, y);
13   cout << "result: " << result << endl;
14   system("pause");
15
16   return 0;
17   }
18   int Func (int fx, int fy)
19   {
20   if ((fx != 0) && (fy != 0))
21    return fx * fy;
22   else
23    return -1;
24   cout << "You will be not here!\n";
25   }
```

Результат

```
Enter x: 10
Enter y: 20
result: 200
```

В программе объявляются три переменные в строках 6 и 7. Первая переменная инициализируется нулем, а вторая и третья –

введенными пользователем значениями. В строке 12 вызывается функция, и в качестве аргумента ей передаются введенные значения. В теле функции происходит сравнение переданных значений с 0. Если они окажутся не равны нулю, то функция вернет произведение этих значений, которое присвоит переменной result. Если же хотя бы одно число равно нулю, то функция возвратит число –1. В строку 24 не попасть никогда.

2.4.5. Значения параметров, используемых по умолчанию

Наряду с обычной формой объявлений функций (заданием прототипа)
float Func (**int** x);
может быть и такая
float Func (**int** x = 70);

Это означает следующее: функция Func ( ) возвращает значение типа **float** и получает параметр типа **int.** Если при ее вызове аргумент представлен не будет, то используются вместо него число 70.

Пример.

```
0     // Значения параметров функции по умолчанию
1     # include <iostream>
2     using namespace std;
3     float Func (int m, int n,
                  float a = 0.3, float b = 0.4);
4     int main()
5     {
6     int m = 15, n = 30;
7     float a = 0, b = 1, p;
8     p = Func (m, n, a, b);
9     cout << "9str  Func: " << p << endl;
10    p = Func (m, n);
11    cout << "11str Func: " << p << endl;
12    system("pause");
13    return 0;
14    }
15    float Func (int m, int n, float a, float b)
16    {
17     return (m - a)/(n + b); }
```

Результат:
9str Func: 0.483871
11str Func: 0.483553

В программе в строке 3 объявлена функция, которая возвращает значение типа **float** и принимает четыре параметра, два из которых заданы по умолчанию. В строках 6 и 7 объявляются и инициализируются целочисленные и вещественные переменные. В строке 8 вызывается функция Func с четырьмя аргументами, результат выполнения функции присваивается переменной p, которая была объявлена в строке 7. В строке 10 опять вызывается функция Func, но с двумя аргументами. Это значит, что третий и четвертый аргументы будут взяты по умолчанию равными 0.3 и 0.4 соответственно (как прописано в объявлении функции).

Ограничение: при объявлении параметров по умолчанию нельзя «умалчивать» левый параметр, если не «умолчен» правый.

2.4.6. Перегрузка функций

В C++ можно создавать несколько различных функций, имеющих одинаковые имена, – так называемая **перегрузка функций**. Подобные функции обязаны отличаться друг от друга **списками параметров** (типом или количеством).
Пример.
int Func (**int**, **int**);
int Func (**long**, **float**);
int Func (**long**);
Говорят, что функция Func перегружена тремя различными списками параметров (тип возвращаемого значения в расчет не берется).

С перегрузкой связывают явление, называемое **полиморфизмом** (существование в программе нескольких перегруженных функций **разного назначения**). Например, функция, усредняющая переданные ей значения аргументов, будет отличаться списком параметров, т. е. типом данных (нужную версию компилятор выберет сам в зависимости от того, какой тип данных передан в функцию для усреднения).

Пример на полиморфизм функций.

```
0    //Полиморфизм функций
1    # include <iostream>
2    using namespace std;
3    int Func (int, int);
4    char Func (char, char, char);
5    int main()
6    {
7    int x = 35, y = 35;
8    char c = 'C', h = '+', a = '+';
9    int resultxy;
10   char resultcha;
11   resultxy = Func (x, y);
12   resultcha = Func (c, h, a);
13   cout <<    "Func (int): " << resultxy << endl;
14   cout <<    "Func (char): " << resultcha << endl;
15   system("pause");
16   return 0;
17   }
18   int Func (int x, int y)
19   {
20    cout << "Func (int) is working." << endl;
21    return x + y;
22   }
23   char Func (char c, char h, char a)
24   {
25    cout << "Func (char) is working." << endl;
26    return char(c + h + a);
27   }
```

Результат:

```
Func (int) is working.
Func (char) is working.
Func (int): 70
Func (char): Щ
```

В программе в строках 3, 4 объявлена перегруженная двумя списками параметров функция Func. В строке 3 она принимает два целочисленных параметра, а в строке 4 – три параметра символьного типа. В строках 18–25 показаны реализации перегруженной функции. Вызов функции осуществляется в строках 11 и 12. В зависимости от того, какого типа аргументы будут переданы в функ-

цию, вызывается та или иная реализация. В реализации функции добавлена строка (20, 25), которая сообщает, какая версия функции сработала (см. результаты программы).

2.4.7. Встраиваемые функции

Вспомним структуру программ на языке C++ (рис. 2.14).

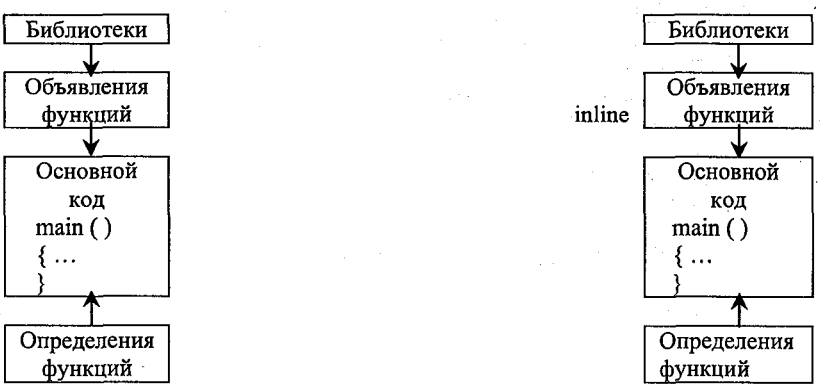


Рис. 2.14. К иллюстрации встраиваемых функций.

Как только в коде программы (т. е. в main ( )) встречается функция Func1 (см. рис. 2.13), производится ее вызов, а потом она ликвидируется. Тем самым из-за такого динамического режима экономится оперативная память, и это хорошо. Но сам процесс вызова и ликвидации функции замедляет выполнение программы (это плохо).

Существует возможность повысить быстродействие: объявить функцию с ключевым словом inline (т. е. объявить ее встраиваемой). Это означает, что компилятор сразу при компиляции вставит функцию в места ее вызова в главном коде (но может и не вставить – сам решает).

Пример.

```
0     // Встраиваемая функция
1     #include <iostream>
2     using namespace std;
3     inline int Multiply (int, int);
4     int main()
5     {
6     int x, y;
7     cout << "Enter x: ";
8     cin >> x;
9     y = x;
10    x = Multiply (x, y);
11    cout << "Result1: " << x << endl;
12    x = Multiply (x, y);
13    cout << "Result2: " << x << endl;
14    x = Multiply (x, y);
15    cout << "Result3: " << x << endl;
16    system("pause");
17    return 0;
18    }
19    int Multiply (int x, int y)
20    {
21     return y * x;
22    }
```

Результат:

```
Enter x: 5
Result1: 25
Result2: 125
Result3: 625
```

В ходе выполнения программы, объявленная в строке 3 функция Multiply вызывается три раза (строки 10, 12, 14), поэтому ее целесообразно было объявить встраиваемой (реализация функции содержит только одну строку), тем самым попытаться повысить быстродействие программы.

2.5. Классы. Объектно-ориентированное программирование (ООП)

2.5.1. Основная идея ООП: создание пользовательского типа – класса

Вспомним, что такое тип переменной, например:

unsigned short int;
беззнаковое короткое целое

За объявлением в программе переменной такого типа стоят, по крайней мере, три вещи:

1. Ее **размер** в оперативной памяти (0–65535, занимает 2 байта).

2. **Тип информации**, которую в ней можно хранить. Например, в данном типе нельзя сохранить имя и фамилию, так как это другой тип – массив символов char (об этом ниже).

3. Указание **операций**, которые могут выполняться с ее участием (например, сложить или присвоить одной из них значение другой).

В языке C++ все типы переменных **встроенные** (в компилятор, среду разработки программ).

Однако на практике возникает необходимость создавать свои **пользовательские типы** данных. И в прототипе языка C++, т. е. в языке C (или Си), такая возможность предусмотрена. Например, с помощью ключевого слова struct (структура):

```
Struct addr
{
char name [30];
unsigned long int zip;
};
```

Данная структура имеет тип addr (адрес). Объявлен новый тип. Но надо объявить еще переменную этого типа:

```
struct addr  addr_info;
```
тип — имя переменной нового типа

Далее с переменной addr_info можно что-то делать, но сильно не разбежишься. Дело в том, что **свои функции** в структуре создавать нельзя. Надо пользоваться, например, библиотечными, а они работают не со структурой в целом, а с ее членами, т. е. с **встроенными** типами.

Этот недостаток (отсутствие методов или функций в структурах) и устранил Страуструп – создатель языка C++. Историческую схему выхода из тупика символически иллюстрирует рис. 2.15.

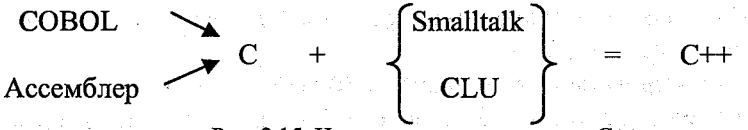


Рис. 2.15. Исторические корни языка C++.

Для эрудиции. Существует понятие «уровень языка» (по близости к «железу»):

COBOL (Pascal, Basic)	–	высокий;
C, C++	–	средний;
Ассемблер	–	низкий.

В язык C был введен новый элемент – «класс» (это структура, struct, но вместе с функциями или методами манипулирования данными).

Класс – это просто пользовательский тип данных. Его объявляет сам программист следующим образом:

```
class River
{
unsigned int itsLength;
unsigned int itsArea;
void RunRiver ( );
};
```

переменные-члены (данные) класса

функция-член (метод) класса

Новый язык (≈ 1985 г.) – так вначале и назывался «C с классами» (C++ – это инкрементное расширение C, часто пишут Си).

Понятие **класс** имеет глубокую философскую нагрузку. Все, что нас окружает (любая вещь), имеет свойства (данные) и методы манипуляции с ними и не только с ними (функции).

Например, река (River) в нашем примере обладает длиной и площадью водосбора, а вода в ней течет.

Еще пример.

У машины есть мотор, колеса, сиденья и т. д. Машина умеет разгоняться, тормозить, возить груз и т. д. Говорят, что в классе **инкапсулированы** данные и методы («закрыты в капсуле»).

Задание (объявление) класса – это задание некой концепции (например, водного объекта), но это еще не сама река, а как бы некий тип. «Переменную» река надо еще объявить:

River Neva; (≡ **int** x)

Neva (т. е. конкретная река) называется **объектом** класса. Это аналог привычного нам понятия «переменная», но не совсем, так как в самом объекте есть свои (например, стандартные или объекты другого класса) «переменные».

Так же как и в случае объявления обычных переменных, объект River надо не только объявить, но и инициализировать, т. е. «наградить» конкретными свойствами: длиной, площадью бассейна и т. п.

Но сначала надо каким-то образом получить доступ к членам класса.

2.5.2. Доступ к членам класса и реализация его методов

Оператор доступа. Чтобы получить доступ к членам объекта класса, используют **оператор доступа**: . (точка):

Neva. itsLength = 74;

(число 74 присваивается переменной-члену itsLength объекта Neva класса River).

Еще пример:

Neva. RunRiver ( );

(вызывается метод RunRiver ( ) объекта Neva).

Важно обратить внимание на то, что значения присваивают объекту, а не классу (ведь класс – это абстракция, т. е. концепция, и ничего ему (ей) присвоить нельзя):

int = 5;	Нельзя	River. itsLength = 74;	Нельзя
int x; x = 5;	Можно	River Neva; Neva.itsLength = 74;	Можно

Закрытые и открытые члены класса. Члены-данные и члены-функции могут быть как закрытыми для внешней среды, так и открытыми, т. е. допускающими возможность своего использования «чужими». Для этого служат два ключевых слова:
public – публичный (открытый);
private – частный (закрытый).

По умолчанию (т. е. если нет никаких слов) члены класса считаются закрытыми:

```
class River
{
int itsLength;
int itsArea;
void RunRiver ( );
};
```

К закрытым членам-переменным могут обращаться только те методы, которые принадлежат тому же классу.
Пример (для класса River).

```
int main ( )
{
River Nil;  // объявляем объект класса River
Nil. itsLength = 6650;
}
```

Пытаемся присвоить его свойству (длине) значение 6650.

Но этого нельзя делать из внешней для River функции main ( ), так как itsLength объявлен закрытым (по умолчанию). Это можно сделать только методом самого класса (если такой метод объявлен).

Ситуацию можно исправить, если этот член объявить открытым:

```
class River
{
  public:
    int itsLength;
    int itsArea;
    void RunRiver ( );
};
int main ( )
{  …
  Nil. itsLength = 6650;
  …. }
```

Действие слова **public** (или **private**) продолжается до того момента, пока не встретится противоположенное слово. Рассмотрим пример.

```
0     // Объявление класса, определение его объекта
1     #include <iostream>
2     using namespace std;
3     class River    //    объявление класса Cat
4     {
5     public:  // переменные будут открытыми
6     int itsLength;
7     int itsArea;
8     };
9     int main()
10    {
11    River Neva;
12    Neva.itsLength = 74; // доступ к переменной-члену
13    cout << "Neva's lenght is " << Neva.itsLength
      << " km.\n";
14    system("pause");
15    return 0;
16    }
```

Результат

```
Neva's length is 74 km.
```

С 3-й строки начинается объявление класса, у которого переменные-члены помещены в раздел открытых переменных и методов класса. Это открывает доступ к переменным-членам из main (строки 12 и 13). (Заметим, что объявление класса заканчивается точкой с запятой.)

Методы доступа к закрытым данным-членам. При использовании классов стремятся оставить его переменные-члены закрытыми. Но при этом в класс вводятся специальные (открытые для внешнего мира) **функции доступа** к этим закрытым данным.

Делается это в основном для того, чтобы во внутренний мир класса (к его данным) не лез кто попало, чтобы способы хранения данных можно было менять и эти изменения не сказывались на внешнем коде самой программы.

Делается для того, чтобы программы на языке C++ носили блочный характер, чтобы их было легко модернизировать, меняя только ее локальные части (чтобы исключить **эмерджентность**).

Объявляются эти методы доступа к закрытым членам класса следующим образом:

```
class River
{
  public:
    int GetLength ( );
    void SetLength (int Length);
    int GetArea ( );
    void SetArea (int Area);
    void RunRiver ( );
  private:
    int itsLength;
    int itsArea;
};
```

Теперь посмотрим, как методы класса определяются (в том числе методы доступа).

Реализация (определение) методов класса. Определение методов класса производится так же, как и глобальных функций (с небольшими нюансами в заголовке). Рассмотрим пример.

```
0     // Реализация  методов класса
1     #include <iostream>
2     using namespace std;
3     class River
4     {
5     public:
6     float GetLength();    //   метод доступа
7     float GetWidth();
8     void SetLength (float length);  //метод доступа
9     void SetWidth(float width);
10    void RunRiver(); //   обычный метод
11    private:// раздел private
12    float itsLength;  //   переменные-члены класса
13    float itsWidth;
14    } ;
15    // Реализации открытых методов доступа GetLength() и GetWidth()
16    float River::GetLength()
17    { return itsLength; }
```

```
18    float River::GetWidth()
19    { return itsWidth; }
20    // Реализации открытых методов доступа SetLength() и SetWidth()
21    void River::SetLength(float length)
22    { itsLength = length; }
23    void River::SetWidth(float width)
24    { itsWidth = width; }
25    // Реализация метода RunRiver()
26    void River::RunRiver ()
27    { std::cout << "The river flows.\n"; }
28    int main()
29    {
30    River Neva;
31    Neva.SetLength(74.0);
32    Neva.RunRiver();
33    cout << "Neva's length is " << Neva.GetLength()
      << " km.\n";
34    River Canal;
35    Canal.SetWidth(6.3);
36    cout << "Canal's width is " << Canal.GetWidth()
      << " m.\n";
37    system("pause");
38    return 0;
39    }
```

Результат

```
The river flows.
Neva's length is 74 km.
Canal's width is 6.3 m.
```

В программе объявлен класс River, в котором объявлено пять методов. Реализация методов описана в строках 16–27. В первой строчке реализации метода указывается тип возвращаемого значения, имя класса, из которого метод, оператор окна видимости (::), имя метода и список параметров.

2.5.3. Конструкторы и деструкторы

Мотивация их появления. Введем два понятия, которые в определенном смысле обобщают то, что вам известно. Когда объявляется переменная, ее надо сразу (или позже) инициализировать, т. е. присвоить какое-либо значение:

```
int Length; // объявляем переменную
...
Length= 7; // присваиваем ей значение (инициализируем)
```

Но можно это сделать и сразу:

```
int Length= 7;
```

Инициализация переменной (т. е. заполнение контейнера или ячейки памяти нужной информацией) гарантирует избавление от предыдущего мусора.

Похожая ситуация возникает и с объектами классов. Объект надо инициализировать. Делается это с помощью специальной функции-члена класса, называемой **конструктором**. Он как бы создает объект класса, т. е. резервирует ячейки памяти и «наклеивает» на них названия переменных-членов.

Объявляя в программе конструктор, надо одновременно объявить еще одну функцию (метод) класса: **деструктор** (он очищает ячейки памяти, выделенные под создаваемый объект, после завершения программы).

Конструкторы бывают двух типов:

1. **Стандартный** (по умолчанию создается самим компилятором, как в предыдущем примере – мы ведь его там даже не упоминали). Он просто резервирует память для объекта:

```
River Volga; // создается объект
River ( ); // конструктор этого объекта (его можно не писать)
```

2. **Пользовательский** – создается программистом в случае, если при создании объекта есть желание в какие-то его ячейки загрузить информацию (например, задать значения членов-данных объекта):

```
River Volga (3530); // создается объект и сразу чем-то инициализируется с помощью специально созданного самим программистом конструктора, который должен быть в программе объявлен и определен
```

Пример использования конструктора и деструктора и два способа инициализации объектов.

Пример.

```
0     //Объявление стандартных
1     // конструктора и деструктора
2     #include <iostream>
3     using namespace std;
4     class River
5     {
6     float itsLength; // private (по умолчанию)
7     float itsWidth;
8     public:
9     River(float initiaiLength, float initialWidth);
      // конструктор
10    ~River();     // деструктор
11    float GetLength();
12    void SetLength(float length);
13    float GetWidth();
14    void SetWidth(float width);
15    void RunRiver();
16    };
17    // Конструктор класса River
18    River::River(float initialLength,
                   float initialWidth)
19    {
20    itsLength = initialLength;
21    itsWidth =  initialWidth;
22    }
23    River::~River()    // деструктор
24    {
25    }
26    float River::GetLength()
27    { return itsLength; }
28    void River::SetLength(float length)
29    { itsLength = length; }
30    float River::GetWidth()
31    { return itsWidth; }
32    void River::SetWidth(float width)
33    { itsWidth = width; }
34    void River::RunRiver()
35    { cout << "The river flows.\n"; }
36    int main()
37    {
38    River Neva(74.0, 1005.4);
```

```
39      Neva.RunRiver();
40      cout << "Neva's length is " << Neva.GetLength()
        << " km.\n";
41      cout << "Neva's width is " << Neva.GetWidth()
        << " m.\n";
42      Neva.RunRiver();
43      Neva.SetLength(75);
44      Neva.SetWidth(999.8);
45      cout << "Now Neva's length is "
        << Neva.GetLength() << " km.\n";
46      cout << "Neva's width is " << Neva.GetWidth()
        << " m.\n";
47      system("pause");
48      return 0;
49      }
```

Результат

```
The river flows.
Neva's length is 74 km.
Neva's width is 1005.4 m.
The river flows.
Now Neva's length is 75 km.
Neva's width is 999.8 m.
```

Эта программа, кроме всего прочего, интересна тем, что иллюстрирует два способа инициализации объекта Neva (его закрытых переменных-членов itsLength и itsWidth):
1) с помощью конструктора (строка 38);
2) с помощью функций доступа SetLength и SetWidth (строки 43, 44).

2.5.4. Постоянные функции-члены

Вспомним две функции доступа из предыдущей программы:

```
void SetLength (int length);
float GetLength ( );
```

Первая функция устанавливает разные значения переменной-члену itsLength:

```
void River :: SetLength (float length)
{
```

```
  itsLength = length;
}
```

т. е. эта функция **меняет** объект.

А функция GetLength ( ) только **показывает** (возвращает) вызываемой функции main ( ) значение itsLength:

```
float River:: GetLength ( )
{
  return itsLength;
}
```

т.е. она с объектом **ничего не делает**. Поэтому считается **хорошим тоном** объявить ее **постоянной**:

```
float GetLength ( ) const;
```

2.5.5. Интерфейс и реализация

Структура программы в C++, клиенты класса и его интерфейс. Вспомним структуру программ в C++ (см. рис. 2.1).

Введение классов структуру программ практически не меняет, но появляются два новых понятия: интерфейс класса и его клиент.

Интерфейсом класса называется его объявление, в котором оговариваются определенные соглашения с теми частями программы, которые будут этот класс использовать (т. е. с **клиентами** класса).

Вспомним, например, объявление класса River:

```
class River
{
 float itsLength;
 float itsWidth;
public:
 River(float initialLength, float initialWidth);
 ~River();
 float GetLength();
 void SetLength(float length);
 float GetWidth();
 void SetWidth(float width);
 void RunRiver();
};
```

интерфейс

В этом объявлении заключены следующие соглашения:
1. Что длина и ширина реки могут быть инициализированы в ее конструкторе; изменены с помощью методов доступа SetLength ( ) и SetWidth ( ); возвращены с помощью методов доступа GetLength ( ) и GetWidth ( );
2. Обещано, что река сможет «бежать» (RunRiver ( ););
3. Подробности реализации (точнее хранения) переменных-членов itsLength и itsWidth **от клиента скрыты** (private), т. е. не являются частью соглашения (это клиента не касается).

Все это и называется интерфейсом.

Типизированность C++. Пример нарушения интерфейса и его последствия. Язык C++ является **типизированным** в том смысле, что если тип данных объявлен не в интерфейсе, то любая попытка программиста его изменить по ходу программы будет пресечена компилятором (он выдаст сообщение об ошибке).

Приведем пример фрагментов программы, в которой нарушен интерфейс.

```
0    // Ошибки компиляции
1    #include <iostream>
2    using namespace std;
3    class River
4    {
5    public:
6    River(int initialLength);
7    ~River() {};
8    int GetLength() const; // постоянный метод доступа
9    private:
10   int itsLength;
11   };
12   // Конструктор класса River
13   River::River(int initialLength)
14   { itsLength = initialLength; }
15   int River::GetLength() const
16   {
17   return (itsLength++);   // нарушение const!
18   }
19   // нарушения интерфейса,
20   // которые приводят к ошибкам компиляции
21   int main()
22   {
```

```
23      River Neva; // не соответствует объявлению конструктора
24      Neva.RunRiver(); // реки "текут", но программист забыл об этом в интер-
        фейсе
25      Neva.itsLength =7; // переменная itsLength закрыта
26      system("pause");
27      return 0;
28      }
```

Расположение объявления класса и реализации его методов. Объявлять класс можно так, как мы это делали до сих пор, т. е. в одном файле с программой (с расширением .cpp), но это не приветствуется.

Рекомендуется (и этого придерживается большинство программистов) помещать объявления в отдельном файле (файле заголовков), имя которого совпадает с именем файла программы, но имеет расширение .h (или .hp, или .hpp). Этот файл заголовков подключается к файлу основного кода так:
include “River.hpp”
(ниже будет рассмотрен пример).

Такая стратегия отделения объявления (интерфейса) от основной программы связана с тем, что клиенту важны возможности класса в целом (а они есть в интерфейсе), а не подробности реализации методов класса. Кроме этого, заголовок может потребоваться и в других файлах с расширением .cpp.

Определения (реализации) методов хранятся в файле с расширением .cpp, т. е. в основном коде и интересуют только программиста.

2.5.6. Методы встраивания реализаций

Так же как и глобальные функции, методы класса можно сделать встраиваемыми (двумя путями):

```
1. inline int River :: GetLength ( )  // с помощью ключевого слова
inline
   {
     return itsLength;
   }
```

2. Если поместить реализацию (определение) функции в объявление класса, то функция становится встраиваемой автоматически.

Ниже будет рассмотрен пример с встроенными методами, а одновременно проиллюстрируем, как объявление класса и его реализация производятся в разных файлах.

Объявляем класс River в файле River.h.

```
0    #include <iostream>
1    using namespace std;
2    class River
3    {
4    float itsLength;
5    float itsWidth;
6    public:
7    River(float initialLength, float initialWidth);
8    ~River();
9    //Следующие методы встроены
10   float GetLength() {return itsLength;}
11   void SetLength(float length) {itsLength = length;}
12   float GetWidth(){return itsWidth;}
13   void SetWidth (float width){itsWidth = width;}
14   void RunRiver() {cout << "The river flows.\n";}
15   };
```

Реализуем (используем) класс River в файле River.cpp.

```
0    //Использование встраиваемых функций
1    // и подключение файлов заголовков
2    #Include "River.h"
3    River::River(float initialLength,
                  float initialWidth)
4    {
5    itsLength = initialLength;
6    itsWidth =  initialWidth;
7    }
8    River::~River()    {}
9    int main()
10   {
11   River Neva(74.0, 1005.4);
12   Neva.RunRiver();
13   cout << "Neva's length is " << Neva.GetLength()
     << " km.\n";
14   cout << "Neva's width is " << Neva.GetWidth()
     << " m.\n";
15   Neva.RunRiver();
```

```
16      Neva.SetLength(75);
17      Neva.SetWidth(999.8);
18      cout << "Now Neva's length is "
19      << Neva.GetLength() << " km.\n";
20      cout << "Neva's width is " << Neva.GetWidth()
        << " m.\n";
21      system("pause");
22      return 0;
23      }
```

Результат

```
The river flows.
Neva's length is 74 km.
Neva's width is 1005.4 m.
The river flows.
Now Neva's length is 75 km.
Neva's width is 999.8 m.
```

В строке 2 подключается файл River.h, в котором содержится объявление класса River со встроенными методами. В файле River.cpp прописаны только реализации конструктора и деструктора. В главной функции программы в строке 11 создается объект типа River, дальше к этому объекту применяются методы.

2.5.7. Классы, содержащие другие классы как данные-члены

Класс – это не просто возможность создавать новый тип. Это сложная вещь, живая. Например, он может иметь наследников, мутировать и размножаться. В своем составе он может иметь другие классы, скажем класс «коробка передач» может входить в класс «автомобиль» и т. д.

Рассмотрим задачу вычисления расстояния между двумя точками, заданных географическими координатами, и определения широты первой точки. Сначала объявим класс «расстояние», в который входит класс «точка» (файл с расширение .hpp: Distance.hpp). А потом создадим исполняемый файл Distance.cpp.

```
0    // Файл Distance.hpp
1    #include <iostream>
2    using namespace std;
3    class Point     // содержит координаты широты и долготы
4    {
5    // используем стандартный конструктор
6    float itsLong;
7    float itsLat;
8    public:
9    void SetLong(float Long) { itsLong = Long; }
10   void SetLat(float Lat) { itsLat = Lat; }
11   float GetLong() const { return itsLong; }
12   float GetLat() const { return itsLat; }
13   };
14
15   class Distance
16   {
17   Point First;
18   Point Second;
19   float  itsLat1;
20   float  itsLat2;
21   float  itsLong1;
22   float  itsLong2;
23   public :
24   Distance (float lat1, float lat2,
          float long1, float long2);
25   ~Distance () {}
26   float GetLat1()  { return itsLat1; }
27   float GetLat2()  { return itsLat2; }
28   float GetLong1() { return itsLong1; }
29   float GetLong2() { return itsLong2; }
30   Point GetFirst() { return First; }
31   Point GetSecond() { return Second; }
32   void SetFirst(Point Location)
33   { First = Location; }
34   void SetSecond(Point Location)
35   { Second = Location; }
36   void SetLat1 (float lat1)  { itsLat1 = lat1; }
37   void SetLat2 (float lat2)  { itsLat2 = lat2; }
38   void SetLong1 (float long1)   { itsLong1 = long1; }
39   void SetLong2 (float long2)   { itsLong2 = long2; }
40   float GetDistance() const;
41   };
```

```
0     // Файл Distance.cpp
1     #include "Distance.hpp"
2     Distance::Distance (float lat1, float lat2,
                          float long1, float long2)
3     {
4     itsLat1 = lat1;
5     itsLat2 = lat2;
6     itsLong1 = long1;
7     itsLong2 = long2;
8     First.SetLat(lat1);
9     Second.SetLat(lat2);
10    First.SetLong (long1);
11    Second.SetLong(long2);
12    }
13    // найти расстояние между точками
14    float Distance::GetDistance() const
15    {
16    float A;
17    float B;
18    cout << "Enter km/min (latitude):  " << endl;
19    cin >> A;
20    cout << "Enter km/min (longitude):  " << endl;
21    cin >> B;
22    return sqrt(pow(((itsLong1-itsLong2)*A),2)+ pow(((itsLat1-itsLat2)*B),2));
23    }
24    int main()
25    {
26    // Инициализировать локальную переменную Distance
27    Distance MyDistance (62.5, 62.0, 31.2, 30.0 );
28    float Dis = MyDistance.GetDistance () ;
29    cout<< "Distance: " << Dis << endl;
30    cout<< "Latitude first point: ";
31    cout<< MyDistance.GetFirst().GetLat()<< endl;
32    system("pause");
33    return 0;
34    }
```

Результат

```
Enter km/min (latitude):
1.867
Enter km/min (longitude):
1.867
Distance: 2.4271
Latitude first point: 62.5
```

В файле Distance.hpp объявлен класс Point, который используется для хранения географических координат (широты и долготы). Внутри класса объявлены две закрытые переменные-члены itsLong и itsLat. В классе объявлены также четыре метода доступа, которые устанавливают и считывают закрытые переменные-члены класса.

В строке 15 начинается объявление класса Distance, который содержит две точки (строки 17, 18). В строке 24 объявлен конструктор, который принимает широту и долготу каждой точки. Реализация конструктора содержится в файле Distance.cpp. В нем происходит инициализация закрытых переменных класса Distance и установка значений двух точек (объектов класса Point).

Помимо методов доступа в классе Distance объявлен метод для вычисления расстояния между точками. Вычисление происходит в строке 22 (файл Distance.cpp).

Файл Distance.cpp начинается с подключения файла Distance.hpp. В главной функции main в строке 27 объявляется и инициализируется объект типа Distance. В строке 28 объявляется переменная, которой будет присвоен результат применения метода GetDistance к объекту MyDistance. В строке 29 выводится значение этой переменной. В строке 31 выводится широта первой точки путем применения к объекту двух методов. Сначала возвращается широта точки (функция GetLat является методом доступа к значению широты в классе Point). Затем следует вызов метода GetFirst, который возвращает объект класса Point. Значение широты этой точки и следует получить. Строка 31 совмещает методы доступа GetFirst и GetLat.

2.6. Циклы

2.6.1. Ветвление и циклы

Ветвление. Вспомним п. 2.3.3, в котором рассматривался процесс ветвления программ: когда код доходил до определенной строки, то в зависимости от условий (указанных в этой строке) программа могла проскакивать через строки (рис. 2.16, *а*, *б*).

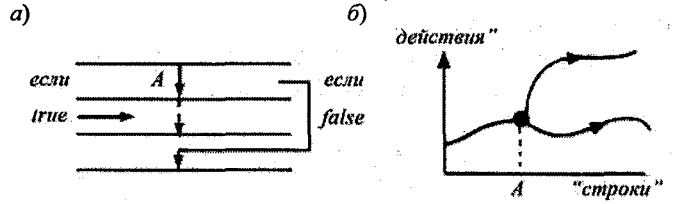


Рис. 2.16. К пояснению «ветвление».

Цикл – это частный случай ветвления, когда точка бифуркации превращается в окружность. Изображающая точка «гуляет» по этой окружности и покидает ее, если выполняется то или иное условие (рис. 2.17, *а*, *б*).

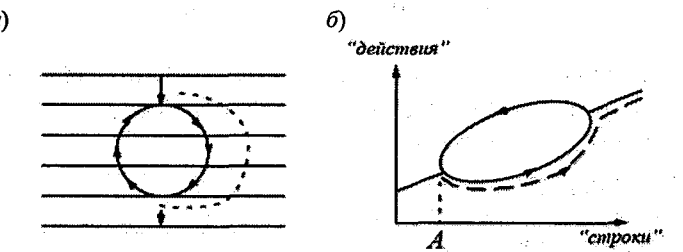


Рис. 2.17. К пояснению «цикл».

Циклы на практике. Когда реально на практике возникает необходимость повторять одни и те же действия? Вспомним сетку, возникающую при реализации численных методов (рис. 2.18).

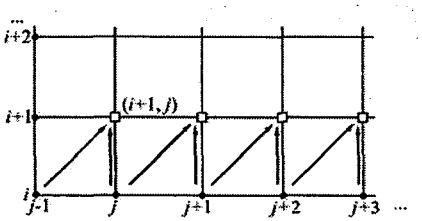


Рис. 2.18. К мотивации использования циклов.

Сначала организуется цикл по i: i, $i+1$, $i+2$, ...
Потом цикл по j: j, $j+1$, $j+2$, ...
Цикл – это повторение одних и тех же действий, например при рекурсии (рис. 2.19, *а*) или при итерации (рис. 2.19, *б*).

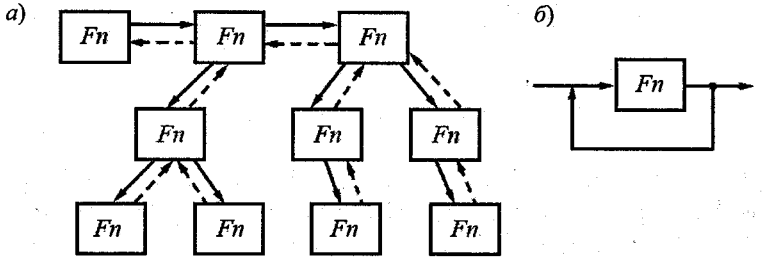


Рис. 2.19. Использование циклов.

Для решения задач организации цикла во многих языках используется оператор цикла goto (безусловного перехода). Рассмотрим пример его использования.

```
0     // Оператор goto
1     # include <iostream>
2     using namespace std;
3     int main()
4     {
5     int i = 3;     // инициализация счетчика
6     label:
7     i ++;     // начало цикла
8     cout << " i: " << i;
9     if (i < 10)
10      goto label; // перейти к началу
11    cout << "\nLast i: " << i << ".\n";
12    system ("pause");
13    return 0;
14    }
```

Результат

```
i: 4 i: 5 i: 6 i: 7 i: 8 i: 9 i: 10
Last i: 10.
```

С помощью оператора goto можно перейти как вперед, так и назад. Неоднократное его использование приводит к запутанной трудно читаемой программе.

Существуют три более приемлемых оператора для организации цикла:

1. **while**;
2. **do** … **while**;
3. **for**.

2.6.2. Цикл while

*«Голый» **while**.* Цикл, организованный с помощью оператора **while**, будет выполняться до тех пор, пока условие в начале цикла остается истинным.

Пример.

```
0     // Оператор while
1     #include <iostream>
2     using namespace std;
3     int main()
4     {
5     int i = 3;
6     while(i < 10) //проверка условия
7     { // тело цикла
8       i ++;
9       cout << " i: " << i;
10    }
11    cout << "\nLast i: " << i << ".\n";
12    system ("pause");
13    return 0;
14    }
```

Результат

```
i: 4 i: 5 i: 6 i: 7 i: 8 i: 9 i: 10
Last i: 10.
```

В строке 5 объявляется и инициализируется счетчик, который будет параметром цикла (строка 6). Цикл выполняется до тех пор, пока значение счетчика будет меньше 10. Увеличение значения переменной i происходит в теле цикла, там же выводится на экран.

Последний раз входим в цикл при значении i равным девяти, в результатах значение десять появляется после выполнения строк 8, 9.

Сложный ***while*** *(с логическим оператором).* Проверяемое в операторе **while** условие может быть очень сложным и содержать логические операторы.
Рассмотрим пример.

```
0    // Сложный while
1    # include  <iostream>
2    using namespace std;
3    typedef unsigned long int ULI;
4    int main()
5    {
6    ULI x;
7    ULI y;
8    cout << "Enter x: ";
9    cin >> x;
10   cout << "Enter y: ";
11   cin >> y;
12   while (x != y && y > 0)
13   {
14    cout << "|";
15    x++;
16    --y;
17   }
18   cout << "\nx: " << x << " y: "<< y << endl;
19   system ("pause");
20   return 0;
21   }
```

Результат

```
Enter x: 20
Enter y: 30
|||||
x: 25 y: 25
```

В программе в строках 5 и 6 объявлены целочисленные беззнаковые длинные переменные. В строках 8–11 у пользователя запрашивается два значения, которые присваиваются переменным. В строке 12 начинается цикл **while** со сложным условием: проверяется значение переменной x, которое не должно равняться значению переменной y, и значение y должно быть больше нуля. В теле цик-

ла (которое будет выполняться пока сложное условие истина) в строке 14 выводится на экран символ, а в строках 15 и 16 происходит изменение переменных x и y. В строке 18 выводятся окончательные значения переменных.

***while** с операторами **continue** и **break**.* Часто возникает необходимость, находясь в теле цикла, либо вернуться в его начало (оператор **continue**), либо выйти из цикла еще до проверки условия продолжения цикла (**break**).

Пример.

```
0    // Операторы break и continue
1    #include <iostream>
2    using namespace std;
3    int main()
4    {
5    int  x1, x2, dx, xx;
6    cout << "Enter a small number: ";
7    cin >> x1;
8    cout << "Enter a large number: ";
9    cin >> x2;
10   if (x1 < x2)
11   {
12   cout << "Enter a step: ";
13   cin >> dx;
14   cout << "Enter a target: ";
15   cin >> xx;
16   while (x1 < 10000 && x2 > 0)
17   {
18    x1+=dx;
19    if (x1 != x2)
20    {
21      x2-=dx;
22      continue;
23    }
24    else
25    {
26     cout << "Target reached!";
27     break;
28    }
29   }
30   }
```

```
31      cout << "\nSmall: " << x1 << " Large: " << x2
        << endl;
32      system ("pause");
33      return 0;
34      }
```

Результат

```
Enter a small number: 2
Enter a large number: 20
Enter a step: 2
Enter a target: 12
Target reached!
Small: 12 Large: 12
```

Это игра. Пользователя просят ввести четыре числа: меньшее (x1), большее (x2), шаг (dx) и конечное значение (xx). За каждый цикл значение переменной x1 увеличивается на шаг, а значение переменной x2 уменьшается в теле if. Игра заканчивается, когда меньшее число будет равно большему. Если такое не произойдет, то игра закончится, когда перестанет выполняться условие в строке 16. Цель игры заключается в том, чтобы угадать конечное число, т. е. угадать равное значение x1 и x2.

while (true). Если значение проверяемого условия задать (или если оно будет получаться) как **true** (истина), то цикл станет бесконечным.

Пример.

```
0       // Цикл while (true)
1       #include <iostream>
2       using std::cout;
3       using std::endl;
4       int main()
5       {
6       int i = 1;
7       while (true)
8       {
9       i = i + 2 * i;
10      if (i > 50) // если убрать эти строчки,
11      break;    // то цикл не завершится
12      }
13      cout << " i: " << i << endl;
```

```
14      system ("pause");
15      return 0;
16      }
```

Результат:
i: 81

В строке 7 оператору **while** назначено условие, которое никогда не станет ложным. Выход из цикла осуществляется с помощью оператора **break** (строка 11). Если убрать строки 10, 11, то цикл станет бесконечным, что приведет к зависанию компьютера. Использовать подобные конструкции следует осторожно, грамотнее было бы организовать проверку значения i в условии продолжения цикла в строке 7.

2.6.3. Цикл do ... while

Оператор **while** обладает той особенностью, что если условие ложно с самого начала, то тело цикла не будет выполнено ни разу. Часто это неудобно, так как желательна какая-то информация из этого тела.

```
0       // Игнорирование тела цикла while
1       // при ложном условии
2       #include <iostream>
3       using namespace std;
4       int main()
5       {
6       int i;
7       cout << "How many waves?: ";
8       cin >> i;
9       while (i > 0)
10      {
11       cout << "Wave!\n";
12       i-=1;
13      }
14      cout << "i is: " << i << endl;
15      system ("pause");
16      return 0;
17      }
```

Результат

```
How many waves?: -2
i is: -2
```

Этот недостаток можно устранить с помощью оператора **do** ... **while**.

Рассмотрим пример.

```
0     // Цикл do ... while
1     #include <iostream>
2     using namespace std;
3     int main()
4     {
5     int i;
6     cout << "How many waves? ";
7     cin >> i;
8     do
9     {
10     cout << "Wave!\n";
11     i-=1;
12    }
13    while (i > 0);
14    cout << "i is: " << i << endl;
15    system ("pause");
16    return 0;
17    }
```

Результат

```
How many waves? -2
Wave!
i is: -3
```

В программе тело цикла выполнится хотя бы одни раз не зависимо от того, какое число введет пользователь. Условие выполнения цикла проверяется в строке 13, т. е. после тела цикла. Если значение переменной i станет меньше нуля, то дальше будет выполняться строка 14.

2.6.4. Цикл for

*Мотивация появления цикла **for** и его связь с циклами **while**.* Если вернуться к циклу **while** и посмотреть, какие **обязательные** действия там надо выполнять, то эти действия такие:

1. Установка начального значения переменных.
2. Контроль истинности условия продолжения цикла.
3. Изменение значений переменной цикла.

Оказывается, что все эти три действия можно совместить в заголовке цикла **for**:

```
for (инициализация счетчика; условие продолжения цикла; приращение счетчика)
{
...
}
```

Рассмотрим пример.

```
0    // Цикл for
1    #include <iostream>
2    using std::cout;
3    int main()
4    {
5    int i;
6    for (i = 3; i > 0; i--)
7    cout << "Privet! "; // тело цикла
8    cout << "\ni: " << i << ".\n" ;
9    system ("pause");
10   return 0;
11   }
```

Результат

```
Privet! Privet! Privet!
i: 0.
```

Инициализация нескольких счетчиков. В цикле может быть несколько счетчиков, как в примере:

```
0    // Несколько счетчиков в цикле for
1    #include <iostream>
2    using std::cout;
3    int main()
4    {
5    for (int i = 0, j = 0; i < 3; i++, j--)
```

```
6     cout << "i: " << i << " j: " << j << "  ";
7     system ("pause");
8     return 0;
9     }
```

Результат

```
i: 0 j: 0  i: 1 j: -1  i: 2 j: -2
```

В программе в строке 5 объявлено и инициализировано два счетчика, но условие продолжения цикла проверяется только по одному параметру. Выполнение двух условий невозможно. В теле цикла выводятся численные значения счетчиков.

Пустые операторы цикла ***for***. В заголовке цикла **for** могут отсутствовать некоторые операторы (вместо них ставятся символы ;).

Если отказаться от первого и третьего оператора, то получим цикл **while**.

Пример.

```
0     // Пустые операторы цикла for
1     #include <iostream>
2     using std::cout;
3     int main()
4     {
5     int i = 10;
6     for ( ; i > 7; )
7     {
8      i --;
9      cout << "Privet!   ";
10    }
11    cout << "\ni: " << i;
12    system ("pause");
13    return 0;
14    }
```

Результат

```
Privet!   Privet!   Privet!
i: 7
```

В данной программе цикл **for** по объявлению похож на цикл **while**; необходимо соблюсти правила организации цикла, т. е. инициализировать счетчик (строка 5), изменять значения счетчика (строка 8).

Пустой цикл ***for***. Так как в заголовке цикла **for** можно размещать много информации, то тело цикла иногда вообще может быть пустым.

Пример.

```
0    // Отсутствие тела цикла у оператора for
1    #include <iostream>
2    using namespace std;
3    int main()
4    {
5    for (int i = 7; i > 0; cout << " i:" << i--);
6    system ("pause");
7    return 0;
8    }
```

Результат

```
i:7 i:6 i:5 i:4 i:3 i:2 i:1
```

Вложенные циклы. Предположим, нам надо заполнить матрицу (таблицу) числами. Это делается с помощью вложенного цикла (цикл в цикле): i меняется от 1 до 3, но при каждом значении i прокручивается цикл по j (рис. 2.20).

Рис. 2.20. Иллюстрация вложенного цикла.

Принцип вложенных циклов заключается в том, что при каждой итерации внешнего цикла внутренний цикл выполняется полностью.

Пример.

```
0     // Вложенный цикл for
1     #include <iostream>
2     using namespace std;
3     int main()
4     {
5     char simbol;
6     cout << "What simbol? ";
7     cin >> simbol;
8     for (int i=0; i<5; i++)
9     {
10     for (int j=0; j<25; j++)
11       cout << simbol;
12     cout << endl;
13    }
14    system ("pause");
15    return 0;
16    }
```

Результат

```
What simbol? o
ooooooooooooooooooooooooo
ooooooooooooooooooooooooo
ooooooooooooooooooooooooo
ooooooooooooooooooooooooo
ooooooooooooooooooooooooo
```

В начале программы пользователь вводит для заполнения матрицы (o). В строке 8 начинается первый цикл **for** (ряды, число которых равняется 5), вложенный цикл (строка 10) – столбцы (25 штук). Заполнение матрицы происходит следующим образом: в первом ряду матрицы (i = 0) заполняется 25 столбцов (строки кода 10,11), затем переходим на второй ряд (строка кода 12, а за ней 8 (i = 1)), опять заполняем 25 столбцов (повторяется вложенный цикл), переходим на третий ряд (строки кода 12, 8 (i = 2)), выводим символ 5 раз (строки кода 10, 11), переходим на четвертый ряд (строки 12, 8 (i = 3)), выводим 25 символов. После того как условие продолжения внешнего цикла **for** перестает выполняться, переходим на строку 14.

2.6.5. Оператор switch

К операторам ветвления программ принадлежит еще оператор **switch**, который может упростить оператор **if**, если есть много вложений.

Его синтаксис:

```
switch (выражение)  // возвращает целочисленное значение
{
 case значение1: оператор;
  break;
 case значение2: оператор;
  break;
 ...
 case значениеN: оператор;
  break;
 default:  оператор;     // если ни одно значение не подходит
}
```

Рассмотрим пример использования оператора **switch**.

```
0     // Оператор switch
1     #include <iostream>
2     using namespace std;
3     int main()
4     {
5     int day;
6     cout << "Enter a number of day: ";
7     cin >> day;
8     switch (day)
9     {
10    case 1:   cout << "Monday."; break;
11    case 2:   cout << "Tuesday."; break;
12    case 3:   cout << "Wednesday."; break;
13    case 4:   cout << "Thursday."; break;
14    case 5:   cout << "Friday."; break;
15    case 6:   cout << "Saturday."; break;
16    case 7:   cout << "Sunday."; break;
17    default:  cout << "Try again.";
18    }
19    system("pause");
20    return 0;
21    }
```

Результат

```
Enter a number of day: 1
Monday.
```

У пользователя запрашивается число. Введенное пользователем значение присваивается переменной day. Если введена 1, то это соответствует оператору case в 10-й строке: выводится сообщение "Monday!" и затем выполняется оператор break, т. е. выходим из оператора switch на строку 19. Оператор break позволяет покидать оператор switch с того места, где он (break) встретился. Если бы в строке 10 отсутствовал оператор break, то на экране появились бы надписи, соответствующие строкам 10 и 11. В 17-й строке предусмотрен случай, когда пользователь вводит число не из диапазона от 1 до 7.

Использование оператора ***switch*** *для создания меню (бесконечный цикл).* Ниже будет приведен пример ветвления и цикла с использованием практически всех известных нам сейчас концепций.

Пример организации меню.

```
0    // Бесконечный цикл для
1    // взаимодействия с пользователем
2    #include <iostream>
3    using namespace std;
4    // объявление функций
5    void read_x();
6    void write_x();
7    int menu();
8    int x = 5;    //глобальная переменная
9    int main()
10   {
11   bool exit = false;
12   for (;;)
13   {
14    int i = menu();
15    switch(i)
16    {
17     case (1): read_x(); break;
18     case (2): write_x(); break;
```

```
19        case (3): break;
20        case (4): exit = true; break;
21        default: cout << "Try again!" << endl;
22        break;
23       }
24       if (exit == true)
25        break;
26      }    // конец бесконечного цикла
27      system("pause");
28      return 0 ;
29      }
30      int menu()
31      {
32       int i;
33       cout << "\n----------------------------------------"  << endl;
34       cout << "1-Read x. 2-Write x. 3-Repeat menu. 4-Quit." << endl;
35       cout << "----------------------------------------"  << endl;
36       cout << ": ";
37       cin >> i;
38       return i;
39      }
40      void read_x()
41      {
42       cout << "x: " << x << endl;
43      }
44      void write_x()
45      {
46       cout << "Enter x: ";
47       cin >> x;
48      }
```

Результат

```
----------------------------------------
1-Read x. 2-Write x. 3-Repeat menu. 4-Quit.
----------------------------------------
: 1
x: 5

----------------------------------------
1-Read x. 2-Write x. 3-Repeat menu. 4-Quit.
----------------------------------------
: 2
Enter x: 10
----------------------------------------
```

```
1-Read x. 2-Write x. 3-Repeat menu. 4-Quit.
----------------------------------------
: 1
x: 10

----------------------------------------
1-Read x. 2-Write x. 3-Repeat menu. 4-Quit.
----------------------------------------
: 3

----------------------------------------
1-Read x. 2-Write x. 3-Repeat menu. 4-Quit.
----------------------------------------
: 4
```

В программе бесконечный цикл начинается в строке 12. В строке 14 происходит вызов функции menu, результат выполнения которой присваивается переменной i. Функция menu выводит на экран пункты меню, и возвращает выбор пользователя (строки 30–39). В строке 15 начинается оператор **switch**, который обрабатывает выбор пользователя. Если пользователь ввел 1, то переходим к оператору **case** (1) (строка 17), и происходит вызов функции `read_x`. После работы функции срабатывает оператор **break**, который прекращает работу **switch**. В строке 24 происходит проверка условия. Если переменная exit равняется true, переходим на строку 25, и бесконечный цикл прекращается, в противном случае снова попадаем на строку 12. Переменная exit будет равняться true только, если пользователь введет цифру четыре, которая соответствует пункту меню Quit.

2.7. Массивы

2.7.1. Массив и доступ к его элементам

Часто возникает необходимость объявлять не один элемент (например, типа **int** или **char**), а набор. **Массив** – это и есть набор элементов для хранения однотипных данных. Массив объявляется так:

int	Array	[13];
тип элементов	имя массива	число элементов

Эта строчка для компьютера означает команду зарезервировать область памяти, достаточную для хранения 13 переменных типа **int** (рис. 2.21).

Доступ к элементам массива организуется так:

Array [7];

Семь – это номер элемента, к которому осуществляется доступ. Тип элементов в массиве не указывается, так как он уже объявлен.

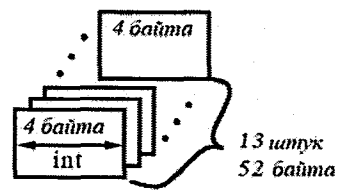


Рис. 2.21. К пояснению массива.

В C++ счет элементов массива начинается с «0», т. е. чтобы обратиться к первому элементу массива, надо написать Array [0]. Поэтому
int Array [13] – это 0, 1, 2, 3, 4, ..., 12.

Рассмотрим пример: объявить массив из пяти целых чисел и заполнить его значениями.

```
0     //Массивы
1     # include <iostream>
2     using namespace std;
3     int main ()
4     {
5     int Array [4];
6     for (int i = 0; i < 4; i++)
7     {
8      cout << "[" << i + 1 << "]: ";
9      cin >> Array [i];
10    }
```

```
11      for (int j = 0; j < 4; j++)
12       cout << j + 1 << ": " << Array [j] << "   ";
13      system("pause");
14      return 0;
15      }
```

Результат

```
[1]: 100
[2]: 200
[3]: 300
[4]: 400
1: 100   2: 200   3: 300   4: 400
```

В программе объявляется массив Array из четырех элементов целочисленного типа. С 6-й строки по 10 с помощью цикла for происходит заполнение массива введенными с клавиатуры пользователем числами. Доступ к элементам массива осуществляется через оператор индекса [ ] (строка 9). Следующий цикл for выводит значения элементов массива. В циклах в качестве индексов массива используются целочисленные переменные i и j.

2.7.2. Проблемы с записью данных

Как могут возникнуть проблемы при записи данных в массив? Предположим объявлен массив (рис. 2.22):
short int Array [6];

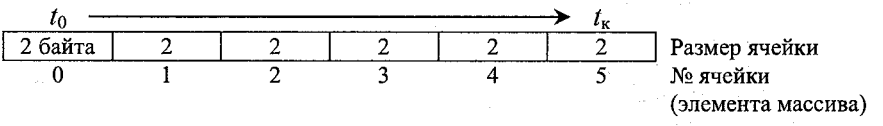


Рис. 2.22. Массив элементов.

Предположим, необходимо записать какое-то значение в переменную массива:
Array [5];
Компилятор делает следующее:

5 * 2 байта = 10 байт (новое смещение).
Указатель смещается на 10 байт в ячейку под номером 5 ($t_к$) и записывает новое значение.

Теперь попробуем записать значение в элемент Array [50] (рис. 2.23). Компилятор, **проигнорировав** тот факт, что такого элемента нет (в объявленном массиве), вычислит смещение:
50 * 2 байта = 100 байт
и запишет туда что-то. Но там может находиться информация и последствия будут непредсказуемыми.

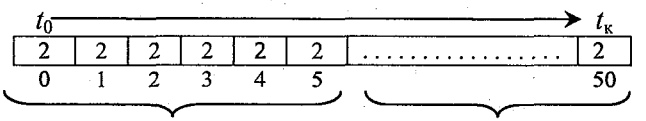


Рис. 2.23. К иллюстрации возможных проблем при записи данных в массив.

Рассмотрим пример записи данных за пределами массива.

```
0   // Проблемы при записи данных за пределами массива
1   #include <iostream>
2   using namespace std;
3   int main()
4   {
5   int i;
6   int Array[25]; // массив для заполнения
7   for (i=0; i<25; i++)
8    Array[i] = 10;
9   cout << "Array[24]: " << Array[24] << "\n\n";
10  cout << "Array[25]: " << Array[25] << "\n\n";
11  Array[25] = 10; //компилятор не сообщит об ошибке
12  cout << "Array[25]: " << Array[25] << "\n\n";
13  system("pause");
14  return 0;
15  }
```

Результат

```
Array[24]: 10

Array[25]: 2147319808

Array[25]: 10
```

В программе объявлен массив в строке 6. В строках 7 и 8 происходит заполнение массива десятками. В строках 9, 10 выводим значения элементов массива с индексами 24 и 25. Но 25-й элемент уже находится за пределами массива, так как последний элемент массива имеет индекс 24. В результате видно, что в ячейке находится какое-то число (мусор), но может находиться и системная информация. В строке 11 происходит запись данных за пределами массива. В строке 12 снова выводится значение с индексом 25. Видно, что содержимое ячейки, не зарезервированной для массива, было заменено. Данная программа может привести к системной ошибке.

Часто говорят об «ошибке последнего столбца». Так как в C++ нумерация элементов массива начинается с «0», то часто забывают, что при int Array [7] массив заканчивается элементом Array [6].

2.7.3. Объявление и инициализация массива

Массив можно инициализировать при объявлении:

int Array [6] = {1, 2, 3, 4, 5, 6};

Можно размер массива и не указывать (компилятор сам поймет из фигурных скобок):

int Array [ ] = {1, 2, 3, 4, 5, 6};

Инициализация элементов за пределами **объявленного** размера массива приведет к ошибке компиляции:

int Array [6] = {1, 2, 3, 4, 5, 6, 7};

Можно инициализировать не все элементы

int Array [6] = {1, 2};

(т. е. объявлено 6, а инициализированы первые два).

Имена массивов и переменных не могут совпадать в пределах одной области видимости. Нельзя одновременно объявить массив myBook [5] и переменную myBook.

Рассмотрим пример программы с объявлением количества элементов массива при помощи перечисления **enum**.

```
0     // Использование перечислений
1     // при задании размера массива
2     #include <iostream>
3     using std::cout;
4     int main()
5     {
6     system("chcp 1251");
7     enum Months {Jan, Feb,  Mar,  Apr, May, Jun, Jul,
8     Aug, Sept, Oct, Nov, Dec, MonthsInYear};
9     int Array[MonthsInYear] = {1,2,3,4,5,6,7,8,9,10,11,12};
10    cout << "Значение сентября: " << Array [Sept]<<"\n";
11    system("pause");
12    return 0;
13    }
```

Результат
Значение сентября: 9

В строке 7 создано перечисление Months. Оно состоит из тринадцати членов: Jan (январь) равно 0, MonthsInYear (количество дней в неделе) – 12. В строке 9 объявлен массив с количеством элементов равным MonthsInYear, т. е. двенадцати. В строке 10 константа перечисления Sept используется в качестве индекса массива. Строка 6 служит для отображения русского алфавита при выводе в строке 10.

2.7.4. Массивы объектов

Элементами массива могут быть и объекты класса

```
class VodnOb
{  ...
};
int main ( )
{
 VodnOb River [5];
...
}
```

Компилятор при объявлении массива объектов определяет (еще не выделяет) размер памяти на основании объявления класса.

Доступ к данным-членам осуществляется в два этапа:
1. [ ] (оператор индекса) – для идентификации элемента массива;
2. . (точечный оператор) – для доступа к определенной переменной-члену: River [i]. SetLength (2 * i);

Рассмотрим пример программы, показывающий создание массива объектов.

```
0     // Массив объектов
1     #include <iostream>
2     using namespace std;
3     class VodnOb
4     {
5     float itsLength;
6     float itsArea;
7     public:
8     VodnOb() { itsLength = 38.7; itsArea = 1500.0; }
9     ~VodnOb() {}
10    float GetLength() const { return itsLength; }
11    float SetArea (float area) { itsArea = area; }
12    float SetLength (int i) {itsLength = 2*sqrt(itsArea * i);}
13    };
14    int main()
15    {
16    VodnOb River[4];
17    for (int i = 1; i < 4; i++)
18     River[i].SetArea (River[i].SetLength(i));
19    for (int i = 1; i < 4; i++)
20     {
21     cout << i << ": " << River[i].GetLength()
      << endl;
22     }
23    system("pause");
24    return 0;
25    }
```

Результат
1: 77.4597
2: 109.545
3: 134.164

В строках 3–13 объявлен класс VodnOb. С 5-й строки начинается раздел закрытых переменных-членов. В строке 8 объявлен и определенен конструктор класса, который при создании объекта класса устанавливает значения закрытых переменных. В строке 9 –

деструктор класса. Далее следуют встроенные методы доступа класса. В строке 16 объявлен массив из четырех объектов. Так как после строки 16 запускается конструктор, определенный в 8 строке, то у всех четырех объектов по умолчанию значения переменных itsLength и itsArea равны соответственно 38.7 и 1500. В цикле for (строка 17) с помощью оператора доступа к каждому объекту массива применяется метод SetArea, аргументом которого является результат применения метода SetLength к объектам массива (устанавливаются новые значения закрытых переменных). В строке 21 в теле цикла выводятся значения закрытых переменных-членов каждого объекта из массива с помощью метода доступа.

2.7.5. Многомерные массивы и их инициализация

Массивы могут иметь любую размерность:
int Array [5] [3] ... [71];
Обычно достаточно двумерного массива.
Рассмотрим для примера шахматную доску (рис. 2.24).

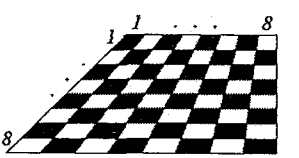


Рис.2.24. Двумерный массив (square Board [8] [8];).

Данные (в клетках) можно представить и одномерным массивом
square Board [64];
но связь с реальностью может быть потеряна. У каждого значения (например, у короля) есть свое место:
Board [0] [3]; // первый ряд, четвертый столбец

Инициализация многомерных массивов происходит следующим образом. Пусть в программе имеется строчка:
int Array [2] [4]; // объявлен двумерный массив
т. е. первые четыре элемента входят в элемент Array [0], следующие четыре – в элемент Array [1].

Инициализация проводится так, что двумерный массив разворачивается построчно:

```
int Array [2] [4] = {
{1, 2, 3, 4},
{4, 5, 6, 7}};
```

Но обычно пишут просто:

```
int Array [2] [4] = {1, 2, 3, 4, 5, 6, 7}
```

Рассмотрим пример программы по созданию многомерного массива.

```
0    // Многомерный массив
1    #include <iostream>
2    using namespace std;
3    int main()
4    {
5    int X[2][3] = { {1,3,5}, {2,4,6}};
6    for (int i = 0; i < 2; i++)
7    {
8     for (int j=0; j < 3; j++)
9      {
10     cout << " X[" << i << "][" << j << "]:" << X[i][j];
11     }
12    cout<<endl;
13   }
14   system("pause");
15   return 0;
16   }
```

Результат

```
X[0][0]:1 X[0][1]:3 X[0][2]:5
X[1][0]:2 X[1][1]:4 X[1][2]:6
```

3. Возможные варианты программ практических заданий с комментариями

3.1. Модель склонового стока с сосредоточенными параметрами I порядка

В качестве исходных данных возьмем ход осадков за 15 суток: 5.1, 15.5, 33.6, 39.2, 28.6, 10.1, 1.00, 1.00, 4.50, 18.2, 20.3, 8.90, 3.50, 2.70, 15.8 и ряд расходов воды за те же 15 суток: 2.23, 4.15, 10.0, 16.1, 20.2, 20.2, 16.8, 8.65, 1.23, 1.51, 3.00, 6.48, 10.2, 9.45, 17.2 (размерность $м^3/с$, хотя это не имеет значения). Хронологические графики приведены на рис. 3.1.

а) *б*)

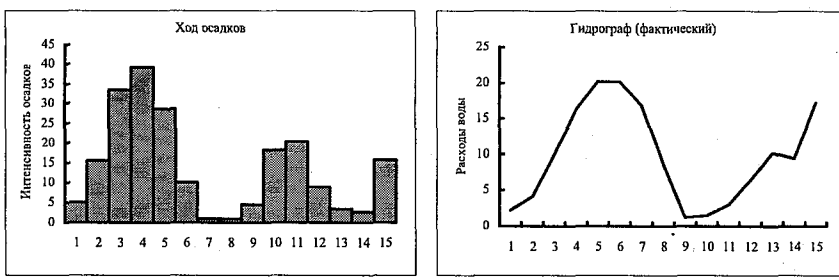


Рис. 3.1. Ход осадков (*а*) и гидрограф (*б*) за 15 суток.

Необходимо провести оптимизацию модели, т. е. найти оптимальные значения коэффициента стока и времени добегания, при которых значение соотношения S/σ_Δ мало, а число оправдавшихся прогнозов P (%) велико.

В листинге 3.1 приведен алгоритм оптимизации и делаются контрольные расчеты при найденных значениях параметров («прогнозы»).

```
1   //3.1. Модель склонового стока с сосредоточенными параметрами I порядка
2   //с оптимизацией коэффициента стока и времени добегания
3   #include <iostream>
4   using namespace std;
5
6   //Объявление переменных
```

```
7   int t, i, j; // счетчики для циклов
8   int dt = 1; // шаг интегрирования
9   float tau, Kst, P; // время добегания, коэффициент стока, число оправ-
    давшихся прогнозов
10  //массив интенсивности осадков
11  float X[15] = {5.10,15.5,33.6,39.2,28.6,10.1,1.00,1.00,4.50,18.2,
    20.3,8.90,3.50,2.70,15.8};
12  //Массив фактических расходов
13  float Q[15] = {2.23,4.15,10.0,16.1,20.2,20.2,16.8,8.65,1.23,1.51,
    3.00,6.48,10.2,9.45,17.2};
14  float Qp[15]; // массив прогнозных расходов воды
15  float Sum, S, delta=0, deltasr, Sums=0, Sigma, Kr, dop; // переменные, необ-
    ходимые для оценки оправдываемости прогноза и эффективности методики
16  float Kr_o=100, P_o=0, Kst_o, tau_o; // оптимальные значения парамет-
    ров
17  char fakt, pr; // символы для изображения гидрографов
18
19  int main()
20  {
21  //Расчет допустимой погрешности
22  for (t=0; t<15; t++)
23  { delta +=(Q[t]-Q[t+1]);}
24  deltasr = delta/14;
25  for (t=0; t<15; t++)
26  { Sums += ((Q[t]-Q[t+1])-deltasr)*((Q[t]-Q[t+1])-deltasr);}
27  Sigma = sqrt(Sums/14);
28  dop = 0.674*Sigma;
29
30  //Прогноз расходов воды с разными параметрами (коэффициентом стока
    и временем добегания)
31  for (tau=1; tau<6; tau+=0.1)
32  {
33  for (Kst=0.1; Kst<1.1; Kst+=0.1)
34  {
35   cout <<"tau "<< tau << "\n" << "Kst "<< Kst << "\n";
36   float opr=0;
37   for (t=1; t<15; t++)
38    {
39     if (t==1)
40       Qp[t]=Q[t-1]+dt*(X[t-1]/tau-Q[t-1]/(Kst*tau));
41     else
42     Qp[t]=Qp[t-1]+dt*(X[t-1]/tau-Qp[t-1] /(Kst*tau));
43     cout << t << ": \t" << Q[t]<<"\t"<< Qp[t]<<"\n";
44     Sum += (Q[t]-Qp[t])*(Q[t]-Qp[t]);
45     if (fabs(Q[t]-Qp[t])<=dop)
46     opr+=1;
47    }
```

```
48   //Критерий эффективности методики
49   S = sqrt(Sum/(14));
50   Sum = 0;
51   Kr = S/Sigma;
52   P = (opr/14)*100;
53   cout << "S/sigma = " << Kr << "\n";
54   cout <<"P "<< P << "\n";
55   //Оптимальные коэф-т стока и время добегания
56   if ((Kr<=Kr_o) && (P>=P_o))
57   { Kr_o=Kr; P_o=P; Kst_o=Kst; tau_o=tau; }
58   }
59   }
60   cout << "\n\nKr_o " << Kr_o << "\n";
61   cout << "P_o " << P_o << "\n";
62   cout << "Kst_o " << Kst_o << "\n";
63   cout << "tau_o " << tau_o << "\n";
64   for (t=1; t<15; t++)
65    {
66    if (t==1)
67      Qp[t]=Q[t-1]+dt*(X[t-1]/tau_o-Q[t-1]/(Kst_o*tau_o));
68    else
69      Qp[t]=Qp[t-1]+dt*(X[t-1]/tau_o-Qp[t-1]/(Kst_o*tau_o));
70    cout << t << ": \t" << Q[t]<<"\t"<< Qp[t]<<"\n";
71   }
72   //Изображение гидрографа фактического и спрогнозированного
73    cout<<"Vvtdite simvol dlya fakta ";
74    cin >>fakt;
75    cout<<"Vvtdite simvol dlya prognoza ";
76    cin>>pr;
77    for(i=0;i<15;i++)
78     {  for (j=0; j<Q[i]; j++)
79         cout<<fakt;
80         cout<<"\n";
81        for (t=0; t<Qp[i]; t++)
82         cout<<pr;
83         cout<<"\n";
84    }
85   system("pause");
86   return 0;
87   }
```

Результат

```
Kr_o 0.857917
P_o 71.4286
Kst_o 0.7
tau_o 3.1
```

```
1:     4.15   2.84751
2:     10     6.5353
3:     16.1   14.3624
4:     20.2   20.3889
5:     20.2   20.2189
6:     16.8   14.1595
7:     8.65   7.95697
8:     1.23   4.61274
9:     1.51   3.93867
10:    3      7.99458
11:    6.48   10.8588
12:    10.2   8.72573
13:    9.45   5.83369
14:    17.2   4.01632
Vvtdite simvol dlya fakta x
Vvtdite simvol dlya prognoza o
xxx

xxxxx
ooo
xxxxxxxxxx
ooooooo
xxxxxxxxxxxxxxxx
ooooooooooooooo
xxxxxxxxxxxxxxxxxxxx
oooooooooooooooooooo
xxxxxxxxxxxxxxxxxxxx
oooooooooooooooooooo
xxxxxxxxxxxxxxxx
ooooooooooooooo
xxxxxxxxx
oooooooo
xx
ooooo
xx
oooo
xxx
oooooooo
xxxxxxx
ooooooooooo
xxxxxxxxxxx
ooooooooo
xxxxxxxxxx
oooooo
xxxxxxxxxxxxxxxxxx
ooooo
```

По результатам работы программы видно, что найденные оптимальное значение для коэффициента стока 0.7 и оптимальное значение для времени добегания 3.1. При таких значениях параметров модели соотношение S/σ_Δ равняется 0.85 и оправдавшихся прогнозов (здесь и ниже слово «прогноз» применяется условно, чтобы подчеркнуть прогностическую направленность моделирования и самой процедуры оптимизации, которая может быть выполнена и по другим критериям) в процентах 72. Ниже в результатах приведены символьные фактический и прогнозный гидрографы, которые следует воспринимать развернутыми на 90° против часовой стрелки (как программно разворачивать графики пояснено ниже).

Прокомментируем листинг программы. В строках с 7 по 17 происходит объявление переменных, некоторые переменные инициализируются. С 19-й строки начинается главная функция программы. С 21-й по 28-ю происходит вычисление среднеквадратического отклонения изменения прогнозируемой величины (Sigma) и допустимой погрешности (dop). С 31-й строки начинается блок программы, в котором происходит определение оптимальных значений параметров модели. Для этого используется три цикла, два из которых являются вложенными друг в друга. Первый цикл перебирает значения времени добегания (tau) с шагом 0.1, второй – коэффициента стока (Kst) также с шагом 0.1, третий цикл организован для расчета прогнозных значений на каждый момент времени. В строках 40 и 42 происходит вычисление прогнозных расходов воды на каждом витке циклов. Отличие этих строк заключается в том, что в строке 40 рассчитывается прогнозный расход в момент времени, равный единице, при этом берется начальное (элемент массива с индексом ноль) значение расхода из фактического ряда. В строке 42 при расчетах используются прогнозные расходы воды за предыдущей момент времени. В строке 44 рассчитывается сумма погрешностей прогноза. В условном операторе в строках 45, 46 подсчитывается число оправдавшихся прогнозов. С 49-й по 52-ю вычисляются критерии эффективности применяемой методики. И в строках 56, 57 выбирается наилучший результат, т. е. оптимальные значения параметров, при которых и S/σ_Δ стремиться к минимальному, и P стремиться к максимальному. В строке 16 эти критерии были инициализированы фиксированными значениями. С 64-й по 71-ю строку дается прогноз с оптимальными значениями

параметров. Результаты выводятся на экран. Последний логический блок посвящен визуальному изображению полученных результатов (строки 73–84).

3.2. Модель склонового стока с сосредоточенными параметрами II порядка

Для расчета по модели склонового стока II порядка используются те же данные, что в модели I порядка (см. п. 3.1). При оптимизации выбираются те значения параметров (условные поверхностное и подземное время добегания, коэффициент стока), при которых соотношение среднеквадратической погрешности к среднеквадратическому отклонению изменения прогнозируемой величины (S/σ_{Δ}) будет минимальным, число оправдавшихся прогнозов (P %) – максимальным.

```
1       //3.2. Модель склонового стока с сосредоточенными параметрами II
        порядка
2       //с оптимизацией времени добегания поверхностного и подземного
        стока, коэффициента стока
3       #include <iostream>
4       using namespace std;
5
6       //Объявление переменных
7       int t, i, j;
8       int dt = 1;
9       float tau, Kst, P, tau1; // добавилось время добегания подземного стока
10      //Интенсивность осадков
11      float X[15] =
        {5.10,15.5,33.6,39.2,28.6,10.1,1.00,1.00,4.50,18.2,20.3,8.90,3.50,2.70,15.8};
12      //Фактический расход
13      float Q[15] =
        {2.23,4.15,10.0,16.1,20.2,20.2,16.8,8.65,1.23,1.51,3.00,6.48,10.2,9.45,17.
        2};
14      float Qp[15];
15      float Sum, S, delta=0, deltasr, Sums=0, Sigma, Kr, dop;
16      float Kr_o=100, P_o=0, Kst_o, tau_o, tau1_o;
17      char fakt, pr;
18
19      int main()
20      {
21      //Расчет допустимой погрешности
22      for (t=0; t<15; t++)
23      { delta +=(Q[t]-Q[t+1]);}
```

```
24      deltasr = delta/14;
25      for (t=0; t<15; t++)
26      { Sums += ((Q[t]-Q[t+1])-deltasr)*((Q[t]-Q[t+1])-deltasr);}
27      Sigma = sqrt(Sums/14);
28      dop = 0.674*Sigma;
29
30      //Прогноз расходов воды с разными параметрами
31      for (tau1=1; tau1<50; tau1+=1)
32      {
33      for (tau=1; tau<16; tau+=1)
34      {
35      for (Kst=0.1; Kst<1.1; Kst+=0.1)
36      {
37       cout <<"tau "<< tau << "\n" << "Kst "<< Kst << "\n";
38       float opr=0;
39       for (t=2; t<15; t++)
40        {
41         if (t==2)
42            Qp[t]=(X[t-1]/tau - Q[t-1]/(Kst*tau) +
        Q[t1]*(2*tau1+(tau1+Kst*tau1)/(Kst*tau)) - tau1*Q[t-2]) /
        (tau1+(tau1+Kst*tau)/(Kst*tau));
43         if (t==3)
44          Qp[t]=(X[t-1]/tau - Qp[t-1]/(Kst*tau) + Qp[t-
        1]*(2*tau1+(tau1+Kst*tau1)/(Kst*tau))- tau1*Q[t-
        2])/(tau1+(tau1+Kst*tau)/(Kst*tau));
45         if (t>3)
46            Qp[t]=(X[t-1]/tau - Qp[t-1]/(Kst*tau) + Qp[t-
        1]*(2*tau1+(tau1+Kst*tau1)/(Kst*tau)) - tau1*Qp[t-
        2])/(tau1+(tau1+Kst*tau)/(Kst*tau));
47         cout << t << ": \t" << Q[t]<<"\t"<< Qp[t]<<"\n";
48         Sum += (Q[t]-Qp[t])*(Q[t]-Qp[t]);
49         if (fabs(Q[t]-Qp[t])<=dop)
50         opr+=1;
51        }
52       //Критерий эффективности методики
53       S = sqrt(Sum/(14));
54       Sum = 0;
55       Kr = S/Sigma;
56       P = (opr/14)*100;
57       cout << "S/sigma = " << Kr << "\n";
58       cout <<"P "<< P << "\n";
59       //Оптимальные коэф-т стока и время добегания
60       if ((Kr<=Kr_o) && (P>=P_o))
61       { Kr_o=Kr; P_o=P; Kst_o=Kst; tau_o=tau; tau1_o=tau1;}
62      }
63      }
64      }
```

```
65      cout << "\n\nKr_o " << Kr_o << "\n";
66      cout << "P_o " << P_o << "\n";
67      cout << "Kst_o " << Kst_o << "\n";
68      cout << "tau_o " << tau_o << "\n";
69      cout << "tau1_o " << tau1_o << "\n";
70
71      for (t=2; t<15; t++)
72       {
73        if (t==2)
74           Qp[t]=(X[t-1]/tau_o - Q[t-1]/(Kst_o*tau_o) + Q[t-
        1]*(2*tau1_o+(tau1_o + Kst_o*tau1_o)/(Kst_o*tau_o))-tau1_o*Q[t-
        2])/(tau1_o+(tau1_o + Kst_o*tau_o)/(Kst_o*tau_o));
75        if (t==3)
76           Qp[t]=(X[t-1]/tau_o - Qp[t-1]/(Kst_o*tau_o) + Qp[t-
        1]*(2*tau1_o+(tau1_o + Kst_o*tau1_o)/(Kst_o*tau_o))-tau1_o*Q[t-
        2])/(tau1_o+(tau1_o + Kst_o*tau_o)/(Kst_o*tau_o));
77        if (t>3)
78           Qp[t]=(X[t-1]/tau_o - Qp[t-1]/(Kst_o*tau_o) + Qp[t-
        1]*(2*tau1_o+(tau1_o + Kst_o*tau1_o)/(Kst_o*tau_o))-tau1_o*Qp[t-
        2])/(tau1_o+(tau1_o + Kst_o*tau_o)/(Kst_o*tau_o));
79        cout << t << ": \t" << Q[t]<<"\t"<< Qp[t]<<"\n";
80       }
81      //Изображение гидрографа фактического и спрогнозированного
82        cout<<"Vvtdite simvol dlya fakta ";
83        cin >>fakt;
84        cout<<"Vvtdite simvol dlya prognoza ";
85        cin>>pr;
86        for(i=0;i<15;i++)
87         {  for (j=0; j<Q[i]; j++)
88            cout<<fakt;
89            cout<<"\n";
90           for (t=0; t<Qp[i]; t++)
91            cout<<pr;
92            cout<<"\n";
93        }
94      system("pause");
95      return 0;
96      }
```

Результат

```
Kr_o 0.793197
P_o 78.5714
Kst_o 0.7
tau_o 1
tau1_o 1
2:    10     7.50167
```

```
3:     16.1    15.1535
4:     20.2    22.5047
5:     20.2    23.6135
6:     16.8    17.0438
7:     8.65    8.3177
8:     1.23    2.59855
9:     1.51    1.16024
10:    3       5.56563
11:    6.48    10.4524
12:    10.2    10.1183
13:    9.45    6.82577
14:    17.2    3.80887
Vvtdite simvol dlya fakta x
Vvtdite simvol dlya prognoza o
xxx

xxxxx

xxxxxxxxxx
oooooooo
xxxxxxxxxxxxxxxx
oooooooooooooooo
xxxxxxxxxxxxxxxxxxxx
oooooooooooooooooooooooo
xxxxxxxxxxxxxxxxxxxx
oooooooooooooooooooooooo
xxxxxxxxxxxxxxxx
oooooooooooooooo
xxxxxxxxx
ooooooooo
xx
ooo
xx
oo
xxx
oooooo
xxxxxxx
ooooooooooo
xxxxxxxxxx
ooooooooooo
xxxxxxxxx
ooooooo
xxxxxxxxxxxxxxxxx
oooo
```

Приведенный листинг отличается от 3.1 тем, что: 1) в переменные добавлено время добегания подземного стока tau1 (строка 9); 2)

производится оптимизация трех параметров – время добегания поверхностного стока, время добегания подземного стока и коэффициент стока (строка 31), добавлен внешний цикл по tau1 с шагом 1; 3) так как рассматриваемая модель второго порядка, то в качестве начального условия берутся фактические расходы в нулевой и в первый моменты времени (условные операторы в строках 41, 43).

3.3. Пример программы для расчета модели склонового стока с грунтовым питанием (с применением классов)

Данный вариант программы отличается использованием класса, в котором хранятся исходные данные, и к программе этот класс подключается через директиву include.

В классе Dan массивы фактических расходов и интенсивности осадков являются закрытыми (выполнение одного из основных принципов языка C++ – инкапсуляции). Инициализация закрытых переменных-членов класса производится с помощью конструктора, в котором объявлены массивы с данными и из которых происходит перезапись значений в «закрытые» массивы. Кроме того, чтобы была возможность добраться до закрытых данных, созданы методы доступа, которые возвращают *i*-й элемент массива расходов воды или осадков.

```
        dan_class.h
1       class Dan // класс с данными
2       {
3       public:
4       Dan ()
5       { // инициализация закрытых переменных
6       float Xck[30] = {5.10,15.5,33.6,39.2,28.6,10.1,1.00,1.00,4.50,
        18.2,20.3,8.90,3.50,2.70,15.8};
7       float Qck[30] =
        {2.23,4.15,10.0,16.1,20.2,20.2,16.8,8.65,1.23,1.51,3.00,6.48,10.2,9.45,17.2
        };
8       for (int j=0; j<30; j++)
9       {
10      Xc[j] = Xck[j];
11      Qc[j] = Qck[j];
12      }
13      }
14      ~Dan() {}
```

```
15      float GetX (int i)  {return Xc[i];}
16      float GetQ (int i)  {return Qc[i];}
17      private:
18      float Xc[30];
19      float Qc[30];
20      };
```

В меженный период у рек преобладает грунтовое питание. В листинге 3.3 рассмотрена ситуация, когда питание реки в межень происходит только за счет грунтовых вод. В модель склонового стока I порядка добавлена грунтовая составляющая, которая объявлена в строке 9. Производится оптимизация трех параметров – времени добегания, коэффициента стока и грунтовой добавки (строки 52–79).

```
1       //3.3. Модель склонового стока с грунтовой добавкой
2       #include <iostream>
3       #include "dan_class.h"
4       using namespace std;
5
6       //Объявление переменных
7       int t, i, j;
8       int dt = 1;
9       float tau, Kst, P, qgr;
10      float X[30];
11      float Q[30];
12      float Qp[30];
13      float Sum, S, delta=0, deltasr, Sums=0, Sigma, Kr, dop;
14      float Kr_o=100, P_o=0, Kst_o, tau_o, qgr_o;
15      char fakt, pr;
16
17      void Multip_gr();
18
19      int main()
20      {
21
22      //получение осадков и расходов
23      Dan XQ;
24      for (i=0; i<30; i++)
25      {
26      X[i]=XQ.GetX(i);
27      Q[i]=XQ.GetQ(i);
28      cout<<X[i]<<"\t"<<Q[i]<<endl; //проверка полученных данных
29      }
30      system("pause");
```

```
31
32      //Расчет допустимой погрешности
33      for (t=0; t<30; t++)
34      { delta +=(Q[t]-Q[t+1]);}
35      deltasr = delta/29;
36      for (t=0; t<30; t++)
37      { Sums += ((Q[t]-Q[t+1])-deltasr)*((Q[t]-Q[t+1])-deltasr);}
38      Sigma = sqrt(Sums/29);
39      dop = 0.674*Sigma;
40
41      Multip_gr();
42
43      system("pause");
44      return 0 ;
45      }
46
47
48      void Multip_gr()
49      {
50      //Прогноз расходов воды
51      //с оптимизацией времени добегания, коэффициента стока и грунто-
        вой составляющей
52      for (qgr=0; qgr<10; qgr+=0.1)
53      {
54      for (tau=0; tau<16; tau+=0.5)
55      {
56      for (Kst=0.1; Kst<1.1; Kst+=0.1)
57      {
58       float opr=0;
59       for (t=1; t<30; t++)
60        {
61         if (t==1)
62            Qp[t]=(X[t-1] - Q[t-1]/Kst + qgr)/tau + Q[t-1];
63
64            Qp[t]=(X[t-1] - Qp[t-1]/Kst + qgr)/tau + Qp[t-1];
65        Sum += (Q[t]-Qp[t])*(Q[t]-Qp[t]);
66        if (abs(Q[t]-Qp[t])<=dop)
67        opr+=1;
68       }
69      //Критерий эффективности методики
70      S = sqrt(Sum/(29));
71      Sum = 0;
72      Kr = S/Sigma;
73      P = (opr/29)*100;
74      //Оптимальные коэф-т стока и время добегания
```

```
75        if ((Kr<=Kr_o) && (P>=P_o))
76        { Kr_o=Kr; P_o=P; Kst_o=Kst; tau_o=tau; qgr_o=qgr;}
77       }
78       }
79       }
80       cout << "\n\nKr_o " << Kr_o << "\n";
81       cout << "P_o " << P_o << "\n";
82       cout << "Kst_o " << Kst_o << "\n";
83       cout << "tau_o " << tau_o << "\n";
84       cout << "qgr_o " << qgr_o << "\n";
85
86       for (t=1; t<30; t++)
87        {
88         if (t==1)
89            Qp[t]=(X[t-1] - Q[t-1]/Kst_o + qgr_o)/tau_o + Q[t-1];
90
91            Qp[t]=(X[t-1] - Qp[t-1]/Kst_o + qgr_o)/tau_o + Qp[t-1];
92            cout << t << ": \t" << Q[t]<<"\t"<< Qp[t]<<"\n";
93       }
94       cout<<"Vvtdite simvol dlya fakta ";
95         cin >>fakt;
96         cout<<"Vvedite simvol dlya prognoza ";
97         cin>>pr;
98         for(i=0;i<30;i++)
99          {  for (j=0; j<Q[i]; j++)
100            cout<<fakt;
101            cout<<"\n";
102           for (t=0; t<Qp[i]; t++)
103            cout<<pr;
104            cout<<"\n";
105        }
106      }
```

Результат

```
Kr_o 0.784543
P_o 75.8621
Kst_o 0.7
tau_o 3
qgr_o 0.9
1:      4.15    2
2:      10      6.51429
3:      16.1    14.9122
4:      20.2    21.1778
5:      20.2    20.9265
6:      16.8    14.6282
7:      8.65    8.2957
8:      1.23    4.9787
```

```
9:      1.51    4.40789
10:     3       8.67556
11:     6.48    11.611
12:     10.2    9.34862
13:     9.45    6.36357
14:     17.2    4.5333
15:     0       7.94125
16:     0       4.4597
17:     0       2.63603
18:     0       1.68078
19:     0       1.18041
20:     0       0.918309
21:     0       0.781019
22:     0       0.709105
23:     0       0.671436
24:     0       0.651705
25:     0       0.641369
26:     0       0.635955
27:     0       0.63312
28:     0       0.631634
29:     0       0.630856
Vvtdite simvol dlya fakta x
Vvedite simvol dlya prognoza o
xxx

xxxxx
oo
xxxxxxxxxx
ooooooo
xxxxxxxxxxxxxxxxx
ooooooooooooooo
xxxxxxxxxxxxxxxxxxxx
oooooooooooooooooooo
xxxxxxxxxxxxxxxxxxxx
oooooooooooooooooooo
xxxxxxxxxxxxxxxx
ooooooooooooooo
xxxxxxxxx
ooooooooo
xx
ooooo
xx
ooooo
xxx
ooooooooo
xxxxxxx
ooooooooooo
```

```
xxxxxxxxxx
oooooooooo
xxxxxxxxx
ooooooo
xxxxxxxxxxxxxxxxx
ooooo
oooooooo
ooooo
ooo
oo
oo
o
o
o
o
o
o
o
o
o
o
```

В листинге 3.3 блок оптимизации и поверочного прогноза сведен в функцию, которая вызывается в главной функции main (строка 41). В строке 17 происходит объявление функции, а в строках с 48 по 106 – ее реализация. Алгоритм, описанный в функции, понятен из комментариев к предыдущим листингам.

В строке 23 объявляется объект типа Dan. На месте объявления объекта компилятор вызывает конструктор, который выделяет память под объект и, в нашем случае, инициализирует переменные-члены класса конкретными значениями. В строках 26, 27 к объекту класса Dan применяются методы доступа, и возвращаемые значения присваиваются элементам массивов для расходов воды и осадков. Строка 28 является не обязательной.

По результатам работы программы видно, что при отсутствии осадков и расходов воды (в массивах с 16-го элемента по 30-й нет значений, т. е. нули) наша река продолжает течь, в ней есть вода за счет питания грунтовыми водами.

3.4. Пример программы для расчета модели кинематической волны

Математическое описание кинематической волны представляет собой модель с распределенными параметрами. Для расчета должны быть заданы не только начальные, но граничные условия. Начальные условия задаются в строке 31 листинга 3.4, в качестве граничных условий выступают расходы воды, находящиеся в файле infile_Q.txt. Из этого файла следует брать значения расходов воды, которые записаны в нем в один столбец. Файловые потоки будут рассмотрены позже (во второй части Практикума), в этом же листинге показан практический алгоритм ввода информации из текстового файла в программу.

Рассматривается участок реки длиной 100 км, на 40-м километре имеется боковой приток с непостоянным расходом, изменяющимся так же, как и расход на границе участка. Кинематическая волна движется со скоростью 20 км/ч. Шаг интегрирования по расстоянию равен 20 км, по времени – 1 ч.

```
0     //3.4. Модель кинематической волны
1     #include <iostream>
2     #include <fstream>
3     using namespace std;
4     //Объявление переменных
5     float Q[10]; //данные для граничных условий
6     float Qr[15][7];
7     int dt=1;
8     int l,t,i; // счетчики для циклов
9     int i1, j, k1, l1; // счетчики для циклов
10    char simvol;
11    float maxum = 0;
12
13
14    class Ruslo
15    {
16    public:
17    Ruslo () { a = 2000; dx=2000; dL=10000; Qb=0;};
18    ~Ruslo() {};
19    void Rez_Ruslo ();
20    private:
21    int a;
22    int dx;
```

```
23    int dL;
24    float Qb; //боковой приток
25    };
26
27    void Ruslo :: Rez_Ruslo ()
28    {
29    cout <<"Kinematics wave\n";
30    for (l=0; l<7; l++)
31      Qr[0][l]=5; //начальные условия
32    for (t=0; t<11; t++)
33     Qr[t][0]=Q[t]; //граничные условия
34    for (t=1; t<11; t++)
35    {
36      cout<<t-1<<"   ";
37      for (l=1; l<7; l++)
38      {
39       if (l==2)
40       Qb=Q[t];
41       Qr[t][l]= Qr[t-1][l]+dt*(Qb-a*(Qr[t-1][l]-Qr[t-1][l-1])/dx);
42       cout << Qr[t-1][l-1]<<"\t";
43       Qb=0;
44       if (Qr[l][t] > maxum)
45       maxum = Qr[t][l];
46      }
47      cout<<endl;
48    }
49    }
50
51    int main()
52    {
53    //получение расходов из файла
54    ifstream fin;
55    fin.open ("infile_Q.txt");
56    for (i=0; i<10; i++)
57    fin >>Q[i];
58    fin.close();
59    Ruslo Msta;
60    Msta.Rez_Ruslo();
61
62    // графическое изображение расходов воды в каждый момент времени
63    // по всей длине участка реки
64    cout<<"Vvedite simvol  ";
65      cin >>simvol;
66    for (k1=0; k1<10; k1++)
67      {
68     for(i1=0;i1<maxum;i1++)
69     {
```

```
70          for (j=0; j<6; j++)
71          {
72          if (i1==(maxum-Qr[k1][j]-1))
73            cout<<Qr[k1][j];
74          if (i1<(maxum-Qr[k1][j]))
75             cout<<"\t";
76          else
77            cout<<simvol<<"\t";
78      }
79      cout<<endl;
80      }
81      cout<<endl<<endl;
82      }
83      system("pause");
84      return 0;
85      }
```

Результат

```
Kinematics wave
0       5       5       5       5       5       5
1       4       5       9       5       5       5
2       10      4       15      9       5       5
3       16      10      20      15      9       5
4       20      16      30      20      15      9
5       20      20      36      30      20      15
6       16      20      36      36      30      20
7       8       16      28      36      36      30
8       1       8       17      28      36      36
9       1       1       9       17      28      36
Vvedite simvol  x

5       5       5       5       5       5
x       x       x       x       x       x
x       x       x       x       x       x
x       x       x       x       x       x
x       x       x       x       x       x
x       x       x       x       x       x

                9
                x
                x
                x
        5       x       5       5       5
4       x       x       x       x       x
x       x       x       x       x       x
x       x       x       x       x       x
```

```
x     x     x     x     x     x
x     x     x     x     x     x

            15
            x
            x
            x
            x
10          x
x           x     9
x           x     x
x           x     x
x           x     x
x           x     x     5     5
x     4     x     x     x     x
x     x     x     x     x     x
x     x     x     x     x     x
x     x     x     x     x     x
x     x     x     x     x     x
            20
            x
            x
            x
16          x
x           x     15
x           x     x
x           x     x
x           x     x
x           x     x
x     10    x     x
x     x     x     x     9
x     x     x     x     x
x     x     x     x     x
x     x     x     x     x
x     x     x     x     x     5
x     x     x     x     x     x
x     x     x     x     x     x
x     x     x     x     x     x
x     x     x     x     x     x
x     x     x     x     x     x
и т.д.
```

В листинге 3.4 в строках 14–25 объявляется класс Ruslo, у которого есть конструктор, деструктор, один метод и четыре зарытых переменных. Реализация метода представлена в строках с 27-й по 49-ю, в методе задаются граничные и начальные условия и рас-

считывается изменение расхода воды по времени и по расстоянию. Для этого организовано два цикла (строки 34, 37), цикл по расстоянию является вложенным. В главной функции в строке 54 объявляется объект fin класса ifstream, в следующей строке к этому объекту применяется встроенный метод open, который связывает объект с текстовым файлом. В цикле (строки 56, 57) происходит считывание чисел из текстового файла и записывание их в массив *Q*. В строке 58 к объекту fin применяется метод close, который закрывает файловый поток. В строке 59 объявляется объект Msta нашего класса Ruslo и ниже к этому объекту применяется метод Rez_Ruslo. После работы метода массив Qr, объявленный в строке 6, заполнится значениями расходов воды на каждый час и на каждый 20-километровый участок. Полученный массив выводится на экран и виден в результатах. Вниз по массиву – это моменты времени (час), вправо – расстояние, т. е. 0-й, 20-й,40-й … 100-й километр. Последний блок в программе посвящен выводу на экран гистограммы на каждый момент времени, на вершинах столбиков стоят значения расходов воды. В результатах приведены не все графики.

3.5. Системная модель процессов водообмена

В листинге 3.5 программно решается задача, описанная в п. 1.3.1. Действия программы понятны из комментариев. Под системной моделью, в данном случае, понимается модель, включающая взаимодействие различных звеньев, участвующих в гидрологическом цикле, таких как склоновый сток, водоем, грунтовые воды, русловой сток.

```
0       // 3.5. Системная модель
1       #include <iostream>
2       using namespace std;
3
4       int l; // счетчик цикла по расстоянию
5       int t; // счетчик цикла по времени
6       int dt= 1; //шаг интегрирования по времени
7
8       int tau= 2; //время добегания
9       float Kst = 0.16; //коэффициент стока
10
11      float Km= 0.46; //морфометрический коэффициент
132
```

```
12    float F = 19.2; // площадь водоема
13    int Q1= 19; // расход левого оттока
14    int Q2= 21 ; // расход правого притока
15
16    int a = 1760; // скорость волны
17    int dx=5000; // шаг по расстоянию для модели руслового стока
18    int dL=20000; //длина реки до моста
19    float Kf =0.1; //коэффициент фильтрации
20    float Hg=1.0; //уровень грунтовых вод
21    int NaUr=1; // начальное наполнение русла
22
23    float X[10] = {200, 500, 500, 300, 150, 0, 0, 0, 0, 0}; //интенсивность
      осадков
24    float Q[10]; //массив расходов воды
25    float H[10]; //массив уровней водоема
26
27    float Hr[5][10]; //массив уровней реки
28    float Hx; // приток к реке
29
30
31    int main()
32    {
33    //модель водосбора
34    cout<<"Vodosbor \n";
35    cout << "Initial charge:  ";
36    cin >> Q[0]; //вводится начальное условие
37    for (t=1; t<10; t++)
38    {
39     Q[t]=Q[t-1]+dt*(Kst*X[t-1]-Q[t-1])/tau;
40     cout << t << ": " << Q[t]<<"\n";
41    }
42
43    //модель водоёма
44    cout<<"\nVodoem  \n";
45    cout << "Initial level:  ";
46    cin >> H[0]; // вводится начальное условие
47    for (t=1; t<10; t++)
48    {
49     H[t]=H[t-1]+dt*((Q[t-1]+Q1-Q2)/F-Km*H[t-1]);
50     cout << t<< ": " << H[t]<<" \n";
51    }
52
53    //модель руслового стока
54    if (a*dt/dx<1) // проверка условия устойчивости
55    {
56    cout << "Initial level:  ";
57    cin >> NaUr; // вводятся начальные условия
```

```
58      for (l=0; l<5; l++)
59        Hr[l][0]=NaUr;
60      for (t=0; t<10; t++)
61        Hr[0][t]=H[t]; // граничные условия
62      for (t=1; t<10; t++)
63      {
64      cout << t<<":  ";
65        for (l=1; l<5; l++)
66        {
67         if (l==2)
68           Hx=0.5;
69         if (l==3)
70           Hx=-Kf *(Hr[3][t-l] - Hg);
71         if (l==4)
72           Hx=0;
73         Hr[l][t]= Hr[l][t-1]+dt*(Hx-a*(Hr[l][t-1]-Hr[l-1][t-1])/dx);
74         cout << Hr[l][t]<<"\t";
75        }
76      cout << "\n";
77      }
78      for (t = 1; t<10; t++)
79       {
80      if (Hr[4][t]>3) //если уровень больше 3 на 20-м км
81        cout << t<< ": Failure!!!\n"; // то – авария!
82       }
83      }
84      else
85      cout << "The condition of stability is not carried out!\n";
86
87      system("pause");
88      return 0;
89      }
```

Результат

```
Vodosbor
Initial charge:  10
1: 21
2: 50.5
3: 65.25
4: 56.625
5: 40.3125
6: 20.1563
7: 10.0781
8: 5.03906
9: 2.51953
```

```
Vodoem
Initial level:  1
1: 0.956667
2: 1.50618
3: 3.33938
4: 5.09754
5: 5.59772
6: 5.01821
7: 3.65547
8: 2.39469
9: 1.45142
Initial level:  1
1: 1          1.5         1.1         1
2: 0.984747   1.824       1.3408      1.0352
3: 1.16829    2.02858     1.51089     1.14277
4: 1.93252    2.22576     1.68312     1.27235
5: 3.0466     2.62254     1.84005     1.41694
6: 3.9446     3.27181     2.06439     1.56587
7: 4.32251    4.00863     2.42109     1.74135
8: 4.08771    4.61912     2.8959      1.98062
9: 3.49177    4.93206     3.39603     2.3028
```

3.6. Системная модель процессов водообмена (программа с использованием классов)

В листинге 3.6 программно с использованием классов решается задача, описанная в п. 1.3.1. Действия программы понятны из комментариев.

Исходная информация:

коэффициент фильтрации $k_ф = 0.1$ 1/ч;

левый приток и правый отток в водоем $Q_л = 19$ м^3/ч; $Q_п = 21$ м^3/ч;

для бассейна верхнего притока время добегания $T_{доб} = 2$ ч;

коэффициент стока $k = 0.16$;

начальное условие для бассейна верхнего притока $Q|_{t_0} = 10$ м^3/ч;

морфометрический коэффициент $k_{морф} = 0.46$ 1/ч;

начальное условие для водоема $H|_{t_0} = 1$ м;

площадь водоема $F = 19.2$ м^2;

скорость распространения волны по руслу $a = \text{const} = 1760$ м/ч;

начальное наполнение русла – 1 м вдоль всей реки;

уровень грунтовых вод $H_{гр} = \text{const} = 1$ м;

метеопрогноз по осадкам на бассейн верхнего притока в водоем:

t ч	0	1	2	3	4	5
$\dot{X}$ м^3/ч	200	500	500	300	150	0

Необходимо рассчитать процесс формирования волны по всем элементам системы.

```
0       // 3.6. Системная модель с использованием классов
1       #include <iostream>
2       using namespace std;
3
4       float X[10] = {200, 500, 500, 300, 150, 0, 0, 0, 0, 0};
5       float Q[10];
6       float H[10];
7       float Hr[5][10];
8       const int dt= 1;
9       int l;
10      int t;
11
12
13      class Vodosbor
14      {
15      public:
16      Vodosbor () { tau = 2; Kst = 0.16;};
17      ~Vodosbor() {};
18      void Rez_vod ();
19      private:
20      int tau;
21      float Kst;
22      };
23
24      class Vodoem
25      {
26      public:
27      Vodoem () {Km=0.46; F=19.2; Q1=19; Q2=21;};
28      ~Vodoem() {};
29      void Rez_vodoem ();
30      private:
31      float Km;
32      float F;
33      int Q1;
34      int Q2;
35      };
36
37      class Ruslo
38      {
```

```
39      public:
40      Ruslo () { a = 1760; dx=5000; dL=20000; Kf =0.1; Hg=1.0; Hx=0;
        NaUr=1;};
41      ~Ruslo() {};
42      void Rez_Ruslo ();
43      private:
44      float NaUr;
45      float a;
46      int dx;
47      int dL;
48      float Kf;
49      float Hg;
50      float Hx;
51      };
52
53      int main()
54      {
55      enum Avaria {MinH=1, SrH, MaxH};
56      //модель водосбора
57      cout<<"Vodosbor \n";
58      Vodosbor Vodosbor_kr;
59      Vodosbor_kr.Rez_vod();
60      //модель водоёма
61      cout<<"\nVodoem  \n";
62      Vodoem Vodoem_kr;
63      Vodoem_kr.Rez_vodoem();
64      //модель руслого стока
65      cout << "\nRuslovoi stok \n";
66      Ruslo Ruslo_kr;
67      Ruslo_kr.Rez_Ruslo();
68       for (t = 1; t<10; t++)
69       {
70        if (Hr[4][t]>MaxH)
71        cout << t<< ": Failure!!!\n";
72        }
73      system("pause");
74      return 0;
75      }
76
77
78      void Vodosbor :: Rez_vod () {
79      cout << "Initial charge:  ";
80      cin >> Q[0];
81      for (t=1; t<10; t++)
82      {
```

```
 Q[t]=Q[t-1]+dt*(Kst*X[t-1]-Q[t-1])/tau;
 cout << t << ": " << Q[t]<<"\n";
}
}


void Vodoem :: Rez_vodoem () {
cout << "Initial level:  ";
cin >> H[0];
for (t=1; t<10; t++)
{
 H[t]=H[t-1]+dt*((Q[t-1]+Q1-Q2)/F-Km*H[t-1]);
 cout << t<< ": " << H[t]<<" \n";
}
}

void Ruslo :: Rez_Ruslo () {
if (a*dt/dx<1)
{
cout << "Initial level:  ";
cin >> NaUr;
for (l=0; l<5; l++)
  Hr[l][0]=NaUr;
for (t=0; t<10; t++)
  Hr[0][t]=H[t];
for (t=1; t<10; t++)
{
cout << t<<":  ";
   for (l=1; l<5; l++)
   {
    if (l==2)
      Hx=0.5;
    if (l==3)
      Hx=-Kf *(Hr[3][t-l]- Hg);
    if (l==4)
      Hx=0;
    Hr[l][t]= Hr[l][t-1]+dt*(Hx-a*(Hr[l][t-1]-Hr[l-1][t-1])/dx);
    cout << Hr[l][t]<<"\t";
   }
cout << "\n";
}
}
else
cout << "The condition of stability is not carried out!\n";
}
```

Результат

Vodosbor
Initial charge: 10
1: 21
2: 50.5
3: 65.25
4: 56.625
5: 40.3125
6: 20.1563
7: 10.0781
8: 5.03906
9: 2.51953

Vodoem
Initial level: 1
1: 0.956667
2: 1.50618
3: 3.33938
4: 5.09754
5: 5.59772
6: 5.01821
7: 3.65547
8: 2.39469
9: 1.45142

Ruslovoi stok
Initial level: 1

1:	1	1.5	1.1	1
2:	0.984747	1.824	1.3408	1.0352
3:	1.16829	2.02858	1.51089	1.14277
4:	1.93252	2.22576	1.68312	1.27235
5:	3.0466	2.62254	1.84005	1.41694
6:	3.9446	3.27181	2.06439	1.56587
7:	4.32251	4.00863	2.42109	1.74135
8:	4.08771	4.61912	2.8959	1.98062
9:	3.49177	4.93206	3.39603	2.3028

3.7. Расчетно-измерительная модель «петли» (уравнение Риккати)

В листинге 3.7 решается задача, описанная в п. 1.4. Действия программы понятны из комментариев.

```
0       // 3.7. Уравнение Риккати
1       #include <iostream>
2       #include <fstream>
3       using namespace std;
4       //Объявление переменных
5       double H[7];
6       double R[7];
7       double I[7];
8       double F[7];
9       double n = 0.03; //коэффициент шероховатости
10      double Q[30]={20}; // начальные условия (начальный расход воды)
11      int alfa = 1;
12      double g = 9.81;
13      double f1[7], f2[7]={0}, f3[7];
14      char fakt;
15      int i, j;
16      double S[7];
17
18      int main()
19      {
20      //получение исходной информации из файла
21      ifstream fin;
22      fin.open ("rikkati.txt");
23      for (i=0; i<7; i++)
24      {
25      fin >>H[i];
26      fin >>R[i];
27      fin >>I[i];
28      fin >>F[i];
29      }
30      fin.close();
31
32      //вычисление f1
33      for(i=0; i<7; i++)
34      {
35      f1[i]=((-1)*g)/(R[i]*F[i]*( ( (pow(R[i], 0.166))/n) *( (pow(R[i], 0.166))/n) ) );
36      }
37      //интерполяция f1 через минуту
38      float f11[30];
```

```
39      float c1;
40      for (i=0;i<7;i++)
41      {
42      c1 = (f1[i+1]-f1[i])/5;
43      for(int j=0;j<6;j++)
44      {
45      int k1 = j+i*5;
46      f11[k1]=f1[i]+j*c1;
47      if (k1==30) {break;}
48      }
49      }
50      //вычисление f2
51      for(i=0; i<7; i++)
52      {
53      f2[i+1]=((2*alfa)/F[i+1])*((F[i+1]-F[i])/300);
54      }
55      //интерполяция f2 через минуту
56      float f22[30];
57      float c2;
58      for (i=0;i<7;i++)
59      {
60      c2 = (f2[i+1]-f2[i])/5;
61      for(int j=0;j<5;j++)
62      {
63      int k2 = j+i*5;
64      f22[k2]=f2[i]+j*c2;
65      if (k2==30) {break;}
66      }
67      }
68      //вычисление f3
69      for(i=0; i<7; i++)
70      {
71      f3[i]= g*I[i]*F[i];
72      }
73      //интерполяция f3 через минуту
74      float f33[30];
75      float c3;
76      for (i=0;i<7;i++)
77      {
78      c3 = (f3[i+1]-f3[i])/5;
79      for(int j=0;j<5;j++)
80      {
81      int k3 = j+i*5;
82      f33[k3]=f3[i]+j*c3;
83      if (k3==30) {break;}
```

```
84      }
85      }
86      //вычисление Q
87      for(i=0; i<31; i++)
88      {
89      Q[i+1]= Q[i] + 60*(f11[i]*Q[i]*Q[i] + f22[i]*Q[i] + f33[i]);
90      cout<<i<<"\t"<<Q[i]<<endl;
91      }
92      //Изображение гидрографа
93        cout<<"Vvtdite simvol dlya gidrografa ";
94         cin >>fakt;
95         for(i=0;i<30;i++)
96          {  for (j=0; j<Q[i]/4; j=j+2)
97             cout<<fakt;
98             cout<<"\n";
99          }
100     system("pause");
101     return 0;
102     }
```

Результат

```
0       20
1       4.6964
2       21.2839
3       37.8841
4       53.0439
5       83.619
6       158.47
7       226.145
8       274.428
9       310.053
10      343.646
11      381.923
12      357.687
13      312.624
14      270.062
15      233.835
16      203.093
17      146.514
18      111.491
19      87.0199
20      68.0741
21      52.4031
22      34.122
23      27.107
24      20.2623
```

25 15.4313
26 10.8515
27 11.3381
28 10.3863
29 9.82418
30 9.16938
Vvtdite simvol dlya gidrografa x
xxx
x
xxx
xxxxx
xxxxxxx
xxxxxxxxxx
xxxxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxxx
xxxxxxxxxxxxx
xxxxxxxxxx
xxxxxxxx
xxxxxx
xxxxx
xxxx
xxx
xx
xx
xx
xx
xx

Заключение

Данное пособие построено таким образом, чтобы одновременно служить Практикумом по дисциплине «Моделирование гидрологических процессов» и неким подобием самоучителя по языку C++. Но именно «подобием», так как подразумевается, что «под рукой» у студента будет преподаватель, к которому всегда можно обратиться за разъяснениями. Следует заметить, что существуют различные варианты того, в какой последовательности надо излагать язык C++ (см. список литературы). В данном учебном пособии мы придерживались в основном логики самоучителя Дж. Либерти и Б. Джонса (разумеется, избегая непосредственного воспроизведения текста или программ). Существуют другие самоучители (например, «от Wiley», см. список литературы), возможно с еще лучшим порядком изложения материала, хотя авторы начинали изучение C++ по самоучителю «Освой самостоятельно C++ за 21 день» и считают его очень хорошим. В любом случае «Практикум» не является в полной мере ни учебником, ни самоучителем по C++. Основным лицом при обучении является преподаватель и именно ему придется компенсировать излишнюю лапидарность раздела, связанного с языком C++.

Предполагается, что в оставшихся двух частях Практикума будут подобным же образом изложены остальные разделы языка C++, а также C++ Builder. С их использованием будут рассматриваться задачи по «Стохастическим моделям» и «Частично инфинитному моделированию».

Список литературы

1. *Коваленко В. В.* Частично инфинитная гидрология. – СПб.: изд. РГГМУ, 2007. – 230 с.
2. *Коваленко В. В.* Теория катастроф и эволюция дифференцируемых многообразий в частично инфинитной гидрологии. – СПб.: изд. РГГМУ, 2008. – 178 с.
3. *Коваленко В. В. , Викторова Н. В., Гайдукова Е. В.* Моделирование гидрологических процессов. 2-е изд., испр. и доп. Учебник. – СПб.: изд. РГГМУ, 2006. – 559 с. (1-е изд.: *Коваленко В. В.* Моделирование гидрологических процессов. – СПб.: Гидрометеоиздат, 1993. – 256 с.).
4. *Культин Н. Б.* Самоучитель C++ Builder. – СПб.: БХВ-Петербург, 2005. – 320 с.
5. *Лафоре Р.* Объектно-ориентированное программирование в C++. Классика Computer Science. 4-е изд. – СПб.: Питер, 2005. – 924 с.
6. *Либерти Дж., Джонс Б.* Освой самостоятельно C++ за 21 день, 5-е изд.: Пер. с англ. – М.: Издательский дом «Вильямс», 2006. – 784 с.
7. *Стивенс Э.* Самоучитель по C++ от Wiley / Пер. с англ.; Под ред. С. М. Молявко. – М.: БИНОМ, Лаборатория знаний, 2005. – 879 с.
8. *Страуструп Б.* Язык программирования C++. Специальное издание. Пер. с англ. – М.: ООО «Бином-Пресс», 2007. – 1104 с.
9. *Шилдт Г.* Полный справочник по C++, 4-е изд.: Пер. с англ. – М.: Издательский дом «Вильямс», 2006. – 800 с.

Оглавление

Table of contens

120-00

Учебное издание

Коваленко Виктор Васильевич
Гайдукова Екатерина Владимировна

ПРАКТИКУМ ПО ДИСЦИПЛИНЕ «МОДЕЛИРОВАНИЕ ГИДРОЛОГИЧЕСКИХ ПРОЦЕССОВ. ЧАСТЬ I. ДИНАМИЧЕСИЕ МОДЕЛИ» (на базе языка С++)

Редакторы Л.В. Ковель, И.Г. Максимова
Компьютерная верстка Н.И. Афанасьевой

ЛР № 020309 от 30.19.96.

Подписано в печать 15.12.10. Формат 60×90 $^{1}/_{16}$. Гарнитура Times New Roman.
Бумага офсетная. Печать офсетная. Усл.-печ. л. 9,3. Тираж 250 экз. Заказ №74/10.
РГГМУ, 195196, Санкт-Петербург, Малоохтинский пр., 98.
ЗАО «НПП «Система», 197045, Санкт-Петербург, Ушаковская наб., 17/1.