Министерство общего и профессионального образования
Российской Федерации

Пермский государственный технический университет

Кафедра автоматизированных систем управления

В. П. Г л а д к о в

КОНСПЕКТ ЛЕКЦИЙ ПО ПРОГРАММИРОВАНИЮ ДЛЯ НАЧИНАЮЩИХ

Рекомендовано учебно-методическим объединением вузов по образованию в области машиностроения и приборостроения в качестве учебного пособия для студентов, обучающихся по специальности 220100

Пермь 1998

ББК Ч23я72
УДК 681.3
Г 52

Гладков В.П. Конспект лекций по программированию для начинающих: Учеб. пособие / Перм. гос. техн. ун-т. - Пермь, 1998. - 217 с.

Изложены основы программирования для студентов младших курсов специальностей АСУ, КРЭС и ЭВТ, читавшегося автором в Пермском государственном техническом университете на протяжении ряда лет. Подробно отражены методы построения алгоритмов на основе их классификации. Приведено большое количество решенных примеров и упражнений для самостоятельной работы и проверки усвоения материала. Пособие может быть использовано учителями средних учебных заведений, лицами, занимающимися самообразованием, любителями информатики.

Ил. 9. Библиогр.: 26 назв.

Рецензенты: зав. кафедрой информатики Пермского государственного педагогического университета, доктор физико-математических наук Е. К. Хеннер, зав. кафедрой математического обеспечения вычислительных систем Пермского государственного университета, доктор физико-математических наук, профессор А. И. Миков, доцент кафедры «Компьютерные системы и сети» МГТУ им. Н.Э.Баумана, кандидат технических наук Г.С. Иванова.

ISBN 5-88151-171-9

ПРЕДИСЛОВИЕ

Использование компьютеров невозможно без знакомства с основами программирования. Однако сложившаяся в настоящее время практика преподавания сводится к изучению какого-либо алгоритмического языка (чаще всего Бейсика или Паскаля), и мало уделяется внимания методам построения алгоритмов.

В предлагаемом учебном пособии, рассчитанном на начинающих, изучение алгоритмического языка (Паскаль) не является главным. В языке выбирается небольшое подмножество (ядро) средств, достаточных для записи простых алгоритмов. Выбор алгоритмического языка Паскаль определяется тем, что он изначально предназначался для обучения программированию, имеет простой, но строгий синтаксис, содержит достаточно средств для записи любого алгоритма. Многолетняя практика использования Паскаля подтвердила его преимущества для воспитания ясного алгоритмического мышления.

Основное внимание в пособии уделяется вопросам построения алгоритмов. К сожалению, не существует алгоритма для построения решения произвольной задачи. Здесь творчество разработчика более чем желательно. Однако можно привести набор эвристических правил, облегчающих построение алгоритма.

Прежде всего рекомендуется придерживаться следующих правил разработки алгоритма:

1. Выявить в постановке задачи используемые понятия, перевести их в соответствующие понятия применяемого алгоритмического языка.

2. Выявить исходные данные и результаты, привести несколько возможных значений исходных данных и соответствующих им результатов.

3. Представить исходные данные и результаты на алгоритмическом языке.

4. Преобразовать «вручную» исходные данные в результат. Выполнить это преобразование для различных значений исходных данных. Записать преобразования, ведущие от исходных данных к результату, на русском языке или рассказать о них кому-либо так, чтобы и он научился выполнять эти преобразования.

5. Записать выявленные преобразования на алгоритмическом языке.

6. Провести трассировку («ручное» выполнение) программы, исправить выявленные ошибки, попытаться упростить решение, сделать его более наглядным.

Эти правила не гарантируют получение работающей программы, но облегчают ее построение.

Построить алгоритм также помогают следующие эвристические правила:

1. Преобразовать исходные данные так, чтобы решение задачи сводилось к уже известному.

2. Разбить исходную задачу на ряд подзадач меньшего объема, решение которых известно или легко может быть получено.

3. Переформулировать условия задачи так, чтобы свести их к знакомой формулировке.

4. Применить известные встроенные функции или их комбинации.

5. Найти алгоритм в учебниках или обобщить известный.

6. Данные одинаковой структуры обрабатывать алгоритмами одинаковой структуры. Выявить структуру исходных данных и вспомнить алгоритм, который ее обрабатывает.

7. Учитывать при построении структуру исходных данных: разные элементы, имеющиеся в данных, вызывают необходимость проверок; повторяющиеся элементы требуют для обработки организации цикла.

8. Если одна и та же работа выполняется в разных частях алгоритма, то ее лучше оформить в виде процедуры или функции.

Все эти и другие правила рассматриваются в учебном пособии на большом количестве примеров. Линейные алгоритмы строятся на основе выявления последовательности действий. Для построения ветвящегося алгоритма следует выявить основные признаки используемых понятий, упорядочить их, например, с помощью таблиц решений, записать в виде программы. Для построения циклов в пособии предлагается классификация циклических алгоритмов. Приводятся схемы записи циклов каждого класса на Паскале. В описываемом подходе следует распознать вид цикла, который потребуется для решения заданной задачи, выбрать подходящую схему и модифицировать ее. Такой же подход используется при работе с массивами.

В учебном пособии рассматривается большое количество примеров, иллюстрирующих используемые понятия и методы. Все приводимые в тексте примеры опробованы и отлажены на компьютере. К наиболее важным темам приводятся упражнения, выполняя которые читатель сможет лучше понять излагаемый материал. Эти примеры рекомендуется выполнять сразу.

Учебное пособие содержит 13 глав. Материал не разбит на отдельные лекции, потому что, в зависимости от состава аудитории, приходится с разной степенью подробности рассматривать разные вопросы. Преподаватель может сделать такое разбиение самостоятельно, а для читателя оно не так и уж важно.

Изложенный материал соответствует курсу лекций «Алгоритмические языки и программирование», который неоднократно читался автором для студентов специальностей «Конструирование радиоэлектронной аппаратуры» (КРЭС), «Автоматизированные системы обработки информации и управления» (АСУ) и «Электронно-вычислительные машины, сети и системы» (ЭВТ).

Автор выражает благодарность Е.А. Кулютниковой за многочисленные замечания, которые по мере своих возможностей он постарался учесть.

1. ВВЕДЕНИЕ

Информатика изучает законы и методы представления, накопления (сбора и хранения), обработки передачи информации. Это целая совокупность наук, объединенных общим объектом, - информацией. В настоящее время во всех информационных процессах: представлении, накоплении, обработке и передаче, существенную роль играют алгоритм и компьютер (ЭВМ), поэтому велико значение умения программировать, т.е. составлять программы для автоматической обработки информации с помощью компьютера.

Каждая наука осуществляет свое познание окружающего мира, образуя понятия об изучаемых ею предметах. Во всяком понятии предмета мыслятся его признаки.

Под признаками понимаются свойства предмета, функции, отношения, действия, т.е. все те его черты, благодаря которым данный предмет есть то, что он есть. Своими признаками предметы или отличаются друг от друга, или сходны друг с другом. Например, размеры предмета (длина, ширина, высота), положение в пространстве, цвет, запах, вкус, то, как он воздействует на другие предметы, - все это признаки предмета.

Понятия образуются следующим образом: выбирается либо произвольное множество объектов, либо множество объектов, заданных для изучения. Далее отбрасываются все индивидуальные признаки, т.е. те признаки, которые принадлежат отдельным объектам данного множества, и удерживаются общие признаки, т.е. те, которые принадлежат всем объектам изучаемого множества. Совокупность этих признаков определяет понятие.

Например, пусть нам неизвестно понятие «собака», и некто собрал и представил нам большое количество собак, тогда сравнивая их между собой, можно установить, что каждая собака обладает большим числом разнообразных признаков.

Например, каждая собака имеет определенный цвет. Однако никакой цвет не входит в понятие «собака», потому что не все собаки имеют один и тот же цвет.

Наличие четырех ног является признаком, общим для всех собак, и, следовательно, входит в понятие «собака».

Каждое понятие представляет абстракцию, т.е. отвлечение от некоторых признаков, без которых, однако, отдельный индивидуум существовать не может. Например, собака как понятие не имеет ни цвета, ни определенной величины, ни породы и т.д., в то время как индивидуальной собаки без этих признаков не существует.

Мы видим, что каждое понятие может быть определено двояко:

1) совокупностью признаков, характеризующих это понятие,

2) совокупностью всех отдельных объектов, входящих в это понятие.

Совокупность всех признаков, характеризующих понятие, называется содержанием понятия. Совокупность всех отдельных объектов, входящих в понятие (т.е. обладающих всеми указанными признаками), называется объемом понятия.

Например, основное содержание понятия «стул» - это соединение четырех признаков:

1) предмет мебели;

2) сидение для одного человека;

3) наличие спинки;

4) отсутствие подлокотников.

В объем понятия «стул» входят все стулья.

Чтобы определить понятие, нет надобности перечислять все признаки, входящие в его содержание, достаточно перечислить лишь существенные признаки. Существенными называются признаки, каждый из которых необходим и которых вместе достаточно для характеристики данного понятия.

Если P1, P2, ..., PN суть существенные признаки некоторого понятия P, то это значит:

1) что, если какой-нибудь отдельный объект не обладает хоть одним из этих признаков, то он не входит в понятие P;

2) что, если какой-нибудь отдельный объект обладает всеми признаками P1, P2, ..., PN, то он входит в понятие P.

Все остальные признаки этого понятия являются следствиями P1, P2, ..., PN, поэтому в определение понятия P следует включать только существенные признаки. Если какой-нибудь предмет не обладает хотя бы одним из существенных признаков, то он не входит в понятие. Если предмет обладает всеми существенными признаками, то он входит в понятие.

Пусть понятие P, имеет признаки P1, P2, ..., PN. В объем этого понятия входят объекты, каждый из которых обладает всеми этими признаками. Увеличим содержание понятия, добавив еще один новый, не являющийся следствием прежних, признак Q. В этом случае не все, а только некоторые объекты, входящие в объем понятия P, обладают признаком Q. Получено новое понятие с признаками P1, P2, ..., PN, Q, его объем составляет часть объема понятия P.

Приведенные рассуждения позволяют сформулировать важный закон логики: если увеличить содержание понятия, то его объем уменьшится, если уменьшить содержание понятия, то объем увеличится.

Возьмем, например, понятие «человек». В его объем входят все люди. Добавим к содержанию этого понятия новый признак: имеющий черную кожу. Тогда объем понятия сократится: теперь в него входят только негры, т.е. часть всех людей.

Продолжая увеличивать содержание понятия, можно довести объем понятия до единичного объекта.

Если объем понятия Q входит как часть в объем понятия P, то понятие P называется родовым, а понятие Q - видовым.

Определение понятия есть раскрытие содержания понятия. Оно устанавливает границу между одним понятием и всеми другими. В логике имеется правило, по которому определение должно содержать только два признака, из которых один должен указывать на ближайшее родовое понятие, его «ближайший род», а другой - на то, чем данное понятие отличается от других понятий, являющихся видами того же рода, - «видовое различие». Например, давая определение квадрата, мы можем в качестве родового понятия выбрать понятие «прямоугольник». Квадрат - это прямоугольник, все стороны которого равны. При этом родовое понятие по объему шире определяемого. В свою очередь, давая определение прямоугольника, опираемся на родовое понятие «параллелограмм». Прямоугольник - это параллелограмм, у которого все углы прямые. Родовое понятие «параллелограмм» по объему шире, чем понятие «прямоугольник». Поступая аналогично, можно определить «параллелограмм», опираясь на более широкое по объему понятие «четырехугольник». Определяя «четырехугольник», опираемся на более широкое понятие «ломаная линия». Этот процесс нельзя продолжать бесконечно долго, т.к. расширяя все время объем родового понятия, мы приходим к предельно широкому понятию, для которого нет родового понятия. Таким понятиям дают описательное определение, перечисляющее все их существенные признаки, при этом явно указывают противоположные понятия или предметы, не входящие в их объем.

К таким основным, предельно широким понятиям информатики относятся понятия: «информация», «исполнитель», «алгоритм», «структура».

2. ИНФОРМАЦИЯ

2.1. Понятие информации

Любое отражение материального мира, которое может быть зафиксировано живым существом или прибором, несет в себе информацию.

Отражение результатов человеческой деятельности или понимания окружающего мира может быть представлено в формальном виде, например в виде наборов букв, цифр или изображений. Такие формальные наборы называются данными. Данные, полученные от источника информации, называются сообщениями. Они становятся информацией в момент использования. Не всяким данным суждено стать информацией. Информацией становятся только те сообщения, которые снимают неопределенность, существующую до их поступления.

Неопределенность можно охарактеризовать количеством возможных выборов действий в конкретной ситуации, а полученную информацию - величиной, на которую уменьшилась степень неопределенности. Например, если необходимо найти дом на известной улице, где живет Иванов, то информация о том, что номер искомого дома четный, уменьшает неопределенность вдвое за счет отброса домов с нечетными номерами. В данном примере степень неопределенности уменьшилась, но не исчезла совсем.

Для численной оценки неопределенности необходимо выбрать функцию, которая зависела бы от количества возможных в данной ситуации выборов. При этом, если возможен только один выбор, то неопределенность равна нулю (функция должна обращаться в нуль, так как неопределенность отсутствует). Например, в ситуации «за весной следует лето» степень неопределенности равна нулю. Кроме того, с возрастанием количества выборов неопределенность ситуации должна увеличиваться. Если имеется две ситуации, то степень неопределенности общей ситуации должна быть равна сумме степеней неопределенности исходных ситуаций.

Искомую функцию в 1928 году предложил американский ученый Р.Хартли: $H=\log_2 N$, где H - степень неопределенности ситуации, N - количество равновозможных выборов в рассматриваемой ситуации.

Если у нас имеется информация о последствиях каждого выбора в данной ситуации, то выбор способа действий будет не случайным. Если такой информации нет, то выбор будет случайным. Случайным называют такое явление, которое может в одних и тех же условиях произойти или не произойти, или произойти как-то иначе. Однако при массовом (неоднократном) повторении случайного явления наблюдаются специфические закономерности, им присущие. Наступление случайного явления количественно оценивается вероятностью. Классическое определение вероятности следующее: вероятностью явления называют отношение числа благоприятствующих случаев к общему числу исключающих друг друга случаев. Поясним это определение на ряде примеров.

Пример 2.1. В ящике лежат 10 одинаковых по форме шаров: 3 белых, 2 синих и 5 красных. Чему равна вероятность того, что наугад вынутый шар окажется красным?

Решение. Так как все 10 шаров одинаковы по форме, то общее число равновозможных случаев равно 10. Число благоприятных случаев равно 5 - по числу красных шаров, следовательно, вероятность вынуть красный шар 5/10=1/2.

Пример 2.2. Бросим две монеты. Какова вероятность того, что на каждой монете выпал герб?

Решение. В этом примере имеется четыре равновероятных события:

1) на первой монете выпала решка, на второй монете - решка;

2) на первой монете - решка, на второй монете - герб;

3) на первой монете - герб, на второй монете - решка;

4) на первой монете - герб, на второй монете - герб.

Из этих четырех равновозможных событий благоприятствующим является только одно - четвертое. Следовательно, искомая вероятность равна 1/4.

Из примера 2.2 следует, что вероятность равновозможного выбора из N $p=1/N$. Тогда формулу Хартли можно переписать: $N=1/p$; $H = \log_2 1/p = -\log_2 p$.

Пример 2.3. Шарик находится в одном из 8 ящиков. Найти неопределенность ситуации.

Решение. $H = \log_2 8 = \log_2 2^3 = 3$.

Пример 2.4. Витязь на распутье. Перед ним три дороги. Найти неопределенность ситуации.

Решение. $H = \log_2 3 = 1{,}585$.

Пример 2.5. Школьник решает, когда учить уроки: днем или вечером? Найти неопределенность ситуации.

Решение. $H = \log_2 2 = 1$.

За единицу измерения неопределенности принимается неопределенность ситуации, имеющей два равновероятных исхода. Такая единица называется бит. В примере 2.3 неопределенность ситуации оценивается в 3 бита, в примере 2.4 - 1,585 бита, в примере 2.5 - 1 бит.

Для определения количества полученной информации достаточно из степени неопределенности исходной ситуации I_1 вычесть степень неопределенности ситуации I_2, в которой оказались после получения информации: $K = I_1 - I_2$.

Пример 2.6. Необходимо найти слово русского языка, содержащее наибольшее количество букв «о».

Решение. Первоначально полем поиска является все множество слов русского языка. Но после получения сообщения о том, что это слово следует искать среди существительных, область поиска уменьшается за счет отбрасывания слов, относящихся к другим частям речи. Таким образом, получена информация, уменьшившая степень нашего незнания.

Если общее количество слов α, то неопределенность можно оценить $\log_2(1/\alpha) = -\log_2\alpha = I_1$. После отбрасывания слов остальных частей речи останутся только существительные. Их количество равно β ($\beta << \alpha$). В этом случае неопределенность можно оценить $\log_2(1/\beta) = -\log_2\beta = I_2$. Количество информации, полученное с этим

сообщением, будет равно уменьшению степени неопределенности $I = I_1 - I_2 = -\log_2\alpha+\log_2\beta$.

Пример 2.7. Пусть имеется колода из 32 игральных карт (в колоде утеряны 4 шестерки). Задумывается одна из карт. Необходимо, задавая вопросы, на которые будут даны ответы «Да», «Нет», угадать задуманную карту.

Решение. Первый вопрос: «Задумана карта черной масти?» На вопрос получен ответ «Нет». Он принес количество информации $K_1 = \log_2 32 - \log_2 16 = 5 - 4 = 1$ бит. Неопределенность ситуации уменьшилась вдвое.

Второй вопрос: «Задумана карта бубновой масти?» Ответ «Да» приносит еще один бит информации. $K_2 = \log_2 16 - \log_2 8 = 4 - 3 = 1$.

Третий вопрос: «Задумана карта-картинка?» Ответ «Нет» уменьшает неопределенность ситуации еще вдвое. $K_3 = \log_2 8 - \log_2 4 = 3 - 2 = 1$. После третьего вопроса осталось 4 варианта: карта может быть 7,8,9 или 10.

Четвертый вопрос: «Задумана семерка или девятка бубновые?» Ответ «Да» приносит еще один бит информации. $K_4 = \log_2 4 - \log_2 2 = 2 - 1 = 1$.

Пятый вопрос проясняет ситуацию: «Задумана семерка бубновая?» Ответ «Нет» дает еще один бит информации. После этого ответа установлено, что задумана девятка бубновой масти.

В 1948 году американский инженер и математик К.Шеннон обобщил формулу Р.Хартли на случай получения информации об одном из N событий, вероятности появления которых различны.

Рассмотрим некоторый умозрительный эксперимент. Пусть имеется генератор, который на своем экране может демонстрировать любую из букв некоторого алфавита, состоящего из К букв. Генерирование осуществляется в соответствии с заданным законом распределения:

A_i	A_1	A_2	..	A_k
P_i	P_1	P_2	..	P_k

Каждая из букв появляется в соответствии с вероятностью появления. За экраном ведется наблюдение: пусть на экране уже появлялось N букв (N - достаточно большое число).

Если мы интересуемся буквой A_i, то она на экране появится приблизительно $N \cdot P_i$ раз. Каждое появление буквы A_i дает $-\log_2 P_i$ бит информации. Всего за все ее появления будет получено $-N \cdot \log_2 P_i$ бит информации.

После демонстрации всех N букв, суммируеем полученную информацию:

$$I = -N \cdot \sum_{i=1}^{k} P_i \cdot \log_2(P_i)$$

На одну букву в среднем приходится $I_{ср} = -\sum_{i=1}^{k} P_i \cdot \log_2(P_i)$ бит информации.

Пример 2.8. Выясним, дома ли наш товарищ, с помощью телефона.

Решение. В этой простой ситуации три альтернативы: либо он дома, либо его нет дома, либо не удалось дозвониться.

Если, позвонив по телефону, мы узнаем, что товарищ дома, т.е. реализуется одна из двух первых альтернатив; то мы получим исчерпывающую информацию, так как будет достигнута полная ясность. Если дозвониться не удалось, то

неопределенность осталась, значит информации получено не было (нулевая информация).

Качество (ценность) получаемой информации связано не только с уменьшением неопределенности, но и со степенью неожиданности полученной информации. Чем более неожиданно сообщение, тем более ценна получаемая информация. Например, если вашего товарища не окажется дома во время школьных занятий, то ценность такой информации не будет очень большой, но ее ценность увеличится, если его не окажется дома ночью или поздно вечером. Качество информации связано также с заинтересованностью в ней человека, получившего сообщение. Так, узнав, что Ивана Иванова, незнакомого вам, не оказалось вечером дома, вы останетесь равнодушным. Если же им окажется ваш одноклассник, с которым вы договорились созвониться вечером, то ценность полученной информации возрастет.

Качество информации связано с последствиями ее получения для вашего поведения. Если эти последствия не требуют изменения поведения, то информация малоценна. Но если, получив сообщение, вам придется, сломя голову, бежать куда-то, чтобы что-то сделать, то ценность полученной информации велика.

Качество информации связано со временем ее получения. Сообщение, полученное сегодня, может заинтересовать вас меньше, чем в том случае, если бы оно было получено вчера. О такого рода ситуациях говорят: «Дорога ложка к обеду».

В технике используют более простой способ измерения информации, который можно назвать объемным. Он основан на подсчете количества символов в сообщении, т.е. связан с его длиной и не учитывает содержание.

Длина различных сообщений зависит от количества символов в алфавите, с помощью которого представляется информация.

Например, одно и то же число восемнадцать в десятичной системе счисления записывается двумя - 18, в двоичной системе счисления - пятью - 10010, в троичной системе счисления - тремя символами - 200.

В вычислительной технике применяют стандартные единицы измерения: бит, байт и производные от них. Бит - это один двоичный символ. Байт - это один символ, который можно представить двоичным кодом. Всего в одном байте можно записать $2^8 = 256$ различных двоичных восьмибитовых последовательностей, или 256 символов.

1024 байта = 1 Кб = 2^{10} байта (Кб - килобайт),
1024 Кб = 1 Мб = 2^{20} байта (Мб - мегабайт),
1024 Мб = 1 Гб = 2^{30} байта (Гб - гигабайт).

Поскольку количество информации, подсчитанное по формуле К.Шеннона, не может быть больше количества двоичных символов в сообщении, то способ измерения информации, принятый в технике, является более грубым.

Упражнения: 1. Разберите решение задачи.

Задача. Библиотечный каталог содержит 265281 карточку. В каждой карточке в среднем 7 строк по 33 символа в строке. Хранение одной карточки обходится в 100 рублей, а хранение одной трехдюймовой дискеты - в 12000 рублей. Определить, где выгоднее хранить каталог - на карточках или на дискетах.

Решение.

На одной карточке в среднем 7·33=231 символ.
На всех карточках хранятся 231·265281=61279911 символов.
Это составляет 61279911/(1024·1024)=58.44 Мб.
Одна трехдюймовая дискета вмещает 1.44 Мб.
Всего потребуется 58.44/1.44=40.58 или 41 дискета.
Для хранения всех дискет потребуется 41·12000=492000 руб.
Для хранения карточек потребуется 26528100 руб.
492000<26528100, поэтому хранить на дискетах выгоднее.

2. Энциклопедия содержит 30 томов. В каждом томе в среднем 350 страниц. На каждой странице в среднем 900 знаков. На хранение одного тома в виде книги тратится десять тысяч рублей, а на хранение одного мегабайта информации в электронном виде - двадцать пять тысяч рублей. Укажите в каком виде выгоднее хранить энциклопедию.

3. Реферативный журнал «Экономика промышленности» содержит в среднем 250 страниц. На каждой странице информация размещается в две колонки по 73 строки. В строке в среднем 46 символов. Всего выходят 12 номеров в год. Сколько пятидюймовых дискет для IBM PC AT потребуется для хранения годовой подписки журнала, если дискета заполняется не более чем на 97%?

4. Путник приехал на распутье. Перед ним 5 дорог. «Вещая птица» подсказала ему, что крайняя справа дорога опасна. Определите количество информации в сообщении птицы.

Абстрактная информация передается с помощью конкретного, зафиксированного на материальном носителе сообщения. Соответствие между ними не является взаимно однозначным. Для одной и той же информации могут существовать разные сообщения, ее передающие, например сообщения на разных языках или добавление к сообщению несущественной (неважной) информации.

Одно и то же сообщение может передавать совершенно разную информацию, если его по-разному интерпретировать. Например, сообщение о двойке за контрольную работу несет разную информацию для ученика, писавшего эту контрольную, для его родителей и для ученика первого класса этой же школы.

Сообщение может восприниматься человеком всеми пятью органами чувств. В соответствии с этим информация делится на виды: зрительную, слуховую, осязательную, вкусовую и обонятельную.

Чаще и больше всего человек воспринимает зрительную информацию. В свою очередь, зрительная информация делится на символьную, числовую и графическую. Информацию, представленную в символьном, числовом и графическом виде, может обрабатывать и компьютер.

2.2. Информационные процессы

Представление, накопление (сбор и хранение), обработку и передачу информации называют информационными процессами. Любой информационный процесс осуществляется с затратами энергии, кроме того информация должна быть зафиксирована на каком-либо материальном носителе, например: бумаге, фотопленке, магнитном носителе, электромагнитных волнах и т.д.

Представление информации на материальном носителе осуществляется с помощью последовательности знаков. Знак - это элемент некоторого конечного множества отличных друг от друга «объектов», набора знаков.

Набор знаков, в котором определен (линейный) порядок, называется алфавитом. Для того чтобы состоящее из знаков сообщение несло информацию, необходимо установить правила придания смысла знакам и их последовательностям, а также правила построения сообщений из знаков.

Знак вместе с его смыслом называется символом. Знаковая система для записи сообщений на основе некоторого алфавита называется языком.

Понять информацию, представленную некоторым сообщением, означает суметь интерпретировать (понять) смысл знаков, из которых оно состоит.

Высказывание «X понимает язык A» выражает тот факт, что лицо X знает правила интерпретации для всех (или по крайней мере для большинства) сообщений, сформулированных на данном языке.

Не всякое представление удобно для обработки, поэтому часто требуется переход от исходного представления информации к представлению, удобному для обработки. Этот процесс называется кодированием информации. Обратный переход называется декодированием. Для кодирования используется кодовая таблица, состоящая из двух столбцов. В первом столбце перечисляются объекты первого представления информации, во втором - объекты второго представления. Объектами могут быть символы алфавита, слова или целые фразы.

Например, задана кодовая таблица

Слово	**Цифра**
Один	1
Два	2
Три	3
Четыре	4
Пять	5
Шесть	6
Семь	7
Восемь	8
Девять	9
Минус	-

Тогда для кодирования фразы «один девять восемь пять минус два один девять» выполняем следующие действия (до тех пор, пока в заданной фразе имеются необработанные слова): выделяем очередное слово, находим его в левом столбце кодовой таблицы и заменяем на цифру, находящуюся в этой же строке в правом столбце таблицы. После кодирования получим 1985-219.

В такой форме представления легче выполнять арифметические операции, а затем полученный результат 1766 можно представить в исходной форме с помощью декодирования.

Для декодирования выбираем цифры по очереди, пока они не кончатся. Каждую выбранную цифру отыскиваем в правом столбце кодовой таблицы и заменяем словом, которое находится в той же строке в левом столбце.

В результате декодирования получаем «один семь шесть шесть». В информатике чаще всего информация кодируется с помощью двух сигналов: включено или выключено, высокое или низкое напряжение и т.д. Принято обозначать одно состояние цифрой 0, а другое цифрой 1. Такое кодирование называется двоичным кодированием, а цифры 0 и 1 именуются битами (от англ. bit - **bi**nary digi**t** - двоичная цифра).

При двоичном кодировании используется последовательность двоичных цифр. Длина последовательности определяется неравенством $m<=2^n$, где m - количество кодируемых объектов, n - длина двоичной последовательности.

При построении кодовой таблицы необходимо учитывать следующее:

1) каждому объекту ставится в соответствие отдельный код;

2) все объекты имеют разные коды;

3) длина двоичной последовательности для каждого кодируемого объекта одинакова.

Пример 2.9. Построить таблицу кодов для кодирования порядкового номера дня недели.

Решение. Всего дней семь, следовательно, из неравенства $7<=2^n$ получаем n=3. Для кодирования можно предложить следующую кодовую таблицу (в этой таблице последовательность 000 не используется):

1	001
2	010
3	011
4	100
5	101
6	110
7	111

Пример 2.10. Построить кодовую таблицу для кодирования порядкового номера планеты солнечной системы.

Решение. Из неравенства $9<=2^n$ находим n=4. Здесь из 16 возможных последовательностей используются только 9. Кодовая таблица может быть такой:

1	1111
2	1110
3	1101
4	1100
5	1011
6	1010
7	1001
8	1000
9	0001

Наиболее часто для кодирования символов в компьютере используется кодовая таблица ASCII. Здесь каждый символ кодируется двоичной последовательностью длиной 8 битов (байтом). Всего в таблице 256 символов.

Долговременные носители позволяют зафиксировать информацию и сохранять ее длительное время. Таким образом, информация может собираться и храниться в специальных хранилищах или на внешних устройствах компьютера.

Такими хранилищами являются библиотеки, фильмотеки, фотоальбомы, архивы и т.д. Для организации хранения необходимо не только правильно отобрать нужную информацию, но и рационально разместить ее, чтобы облегчить поиск. Бессистемное, хаотичное размещение информации допустимо только для хранилищ небольшого объема. В больших хранилищах информация упорядочивается по какому-либо признаку. Например, в телефонных справочниках и словарях она упорядочивается по алфавиту. В небольшой школьной библиотеке можно упорядочить литературу по изучаемым дисциплинам.

Если упорядоченности нет, то при поиске придется просматривать все книги подряд, пока не встретится нужная. В случае отсутствия книги придется просмотреть всю библиотеку.

Если книги упорядочены по алфавиту и нам необходима, например, книга, начинающаяся на «Ю», то поиск мы будем вести сразу в том месте, где стоят книги, начинающиеся на «Ю».

В более сложных случаях для облегчения поиска используются индексы. Примером индекса является оглавление какой-либо книги. Обычно индекс состоит из двух столбцов. В первом столбце, упорядоченном, например, по алфавиту, перечисляются значения поискового признака, во втором указываются номера страниц, на которых описывается соответствующее значение поискового признака.

Еще одним способом облегчения поиска является составление инвертированного списка.

Часто после безуспешных поисков, можно услышать фразу: «Записывать надо». Вот такую запись, которая указывает в каких местах лежит интересующий нас предмет, и называют инвертированным списком.

Примеры инвертированного списка:

1. Простой карандаш лежит:
• в пенале в портфеле;
• на третьей книжной полке, справа;
• во втором левом ящике письменного стола.
2. График функции секанс (y=1/cos(x)) можно найти в книгах:

• Н.А. Виргенко и др. Графики функций. - Киев: Наукова думка,1979. С. 123-124.

• И.Н. Бронштейн, К.А. Семендяев. Справочник по высшей математике для инженеров и учащихся втузов - М: Наука,1986. С. 118.

Во время хранения информация должна быть защищена от порчи. Кроме того, информация устаревает, ее необходимо своевременно обновлять.

Хранящаяся информация может выбираться по разным критериям, систематизироваться, выдаваться по запросам, преобразовываться из одного вида в другой. Эти процессы называются обработкой информации. Важно понять, что эта обработка никогда не добавляет информации. Она состоит в извлечении интересующей информации, ее перегруппировке, преобразовании, классификации и т.д. Конечно, обработка (переработка) информации, как всякий процесс преобразования, позволяет получить на выходе нечто отличное от того, что

подавалось на вход. Например, четверка на выходе калькулятора отличается от «2·2=», поданного на вход. Однако это не новая информация, а лишь другая форма представления того, что было на входе.

Обработанная и удачно представленная информация помогает человеку принять правильное решение. Больше того, она становится значимой для человека в момент использования, когда уменьшается степень неопределенности, уменьшается его незнание. В качестве примера, того, как обработанная информация облегчает принятие верного решения, рассмотрим ситуацию, когда командующий армией должен принять решение о наступлении. Если в этот момент он будет думать о каждой пуговице на мундирах солдат, о каждой портянке, то вряд ли ему удастся принять верное решение. Для принятия правильного решения всю имеющуюся информацию надо обработать, т.е. штаб должен свести солдат в роты, роты объединить в полки, отметить их размещение на карте, указать численность и размещение неприятеля и предоставить эти сведения командующему. В этом случае его решение будет обоснованным.

Важным информационным процессом является передача информации от человека к человеку, от одного поколения к другому. Наиболее известны способы передачи информации с помощью долговременных (газеты, журналы, книги, аудио- и видеокассеты, пластинки, фото- и кинопленки) и недолговременных (специальные физические устройства: телевидение, радио, телефон, телеграф, телетайп, компьютерные сети и другие устройства связи) носителей. Передача информации происходит во времени, поэтому в качестве носителей используют такие физические величины, которые могут изменяться во времени. Изменение некоторой физической величины во времени, обеспечивающее передачу сообщения, называется сигналом. Та характеристика сигнала (длительность, частота, яркость, цвет и т.п.), которая служит для представления сообщения, называется параметром сигнала. Сигнал называется дискретным, если параметр сигнала может принимать лишь конечное количество значений.

В передаче информации всегда участвуют две стороны: тот, кто передает информацию, - источник, и тот, кто ее получает, - приемник. Информация от источника к приемнику передается по каналу связи. В канале связи на передаваемую информацию могут наложиться помехи и исказить ее. Для избавления от этого нежелательного явления передачу информации дублируют, преобразуют специальным образом, чтобы повысить ее помехозащищенность. В канале связи может произойти утечка информации (несанкционированный доступ). Для его предотвращения информацию шифруют, сжимают ее объем, чтобы увеличить скорость передачи.

3. ИСПОЛНИТЕЛИ. КОМПЬЮТЕР - УНИВЕРСАЛЬНЫЙ ИСПОЛНИТЕЛЬ

3.1. Понятие «Исполнитель. Структура компьютера

Исполнитель - это человек, организация или техническое устройство, умеющие исполнять набор команд и совершать ряд проверок. Эти команды и проверки называются системой команд исполнителя (СКИ). Команда обычно состоит из

указания данных, которые нужно обработать, и приказа как это сделать. Результатом выполнения команды являются обработанные данные, изменение состояния окружающей среды или изменение внутреннего состояния исполнителя. С помощью проверок определяют состояние окружающей среды или внутреннее состояние исполнителя, пригодность представленных для обработки данных. В результате проверки вырабатывается логическое значение: истина или ложь.

Как правило, исполнитель может работать только с одним видом информации. Например, калькулятор работает с числовой информацией, фотоувеличитель - с графической, магнитофон - со звуковой, пишущая машинка - с текстовой. Компьютер может работать с любой информацией: с символьной, числовой, графической, звуковой. Набор его команд универсален и может использоваться для реализации любого алгоритма, поэтому компьютер называют универсальным исполнителем алгоритмов.

Центральную часть компьютера составляют процессор и оперативная память. Их работу координирует устройство управления. Процессор и устройство управления вместе называются центральным процессором.

Центральный процессор исполняет все команды, которые входят в СКИ компьютера, и характеризуется своим быстродействием, т.е. способностью выполнять определенное количество команд в единицу времени. Чем выше быстродействие, тем выше класс компьютера, тем быстрее и более сложные задачи он решает.

Оперативная память предназначена для хранения данных и программ их обработки. Хранящаяся в ней программа управляет работой компьютера, т.е. указывает, какие команды и в каком порядке необходимо выполнить, чтобы получить результат. Оперативная память является энергозависимым устройством, т.е. информация в ней сохраняется до тех пор, пока компьютер подключен к электрической сети. Оперативная память состоит из большого количества ячеек, в каждую из которых можно записать порцию информации. Величина этой порции может изменяться для разных компьютеров. В современных компьютерах чаще всего размер ячейки равен одному байту. Каждая ячейка имеет свой адрес (порядковый номер), по которому можно ее найти. Содержимое ячейки можно читать сколько угодно раз, при этом информация, хранящаяся в ячейке, сохраняется. При записи новой информации в ячейку старая информация теряется. Новая информация может получиться в результате выполнения каких-либо команд центральным процессором, пересылки данных из одной ячейки памяти в другую или получения информации извне. Оперативная память характеризуется объемом, т.е. количеством имеющихся ячеек, скоростью поиска (извлечения) и скоростью записи информации в ячейку. Поскольку эта память дорогая, ее объем сравнительно невелик.

Для работы компьютера этих устройств достаточно, но для общения с человеком нужны еще дополнительные устройства.

Устройства ввода предназначены для получения программ и данных от человека, преобразования их в вид, удобный для обработки и размещения их в оперативной памяти. Современные устройства ввода - клавиатура, «мышь», световое перо, сканер, джойстик, трекбол, дигитайзер и т.д.

Устройства вывода предназначены для преобразования полученных результатов в вид, удобный для восприятия человеком и для получения их твердой копии. К устройствам вывода относятся принтер, монитор, графопостроитель, динамик и т.д.

Для долговременного хранения больших объемов информации у компьютера имеется внешняя память (внешние запоминающие устройства). Это сравнительно недорогие энергонезависимые запоминающие устройства большого объема. Наиболее распространены дисковые запоминающие устройства: стационарные - винчестеры, сменные - дискеты. Стримеры - устройства хранения информации на магнитной ленте - характеризуются большими объемами хранимой информации, но низкой скоростью доступа. Имеются оптические запоминающие устройства большого объема.

3.2. Работа компьютера

Для начала работы компьютера необходимо в оперативную память загрузить программу и исходные данные. Далее работой компьютера управляет программа. Исполнение программы осуществляется с помощью следующей последовательности шагов:

1. Выбирается очередная команда программы и пересылается из оперативной памяти в центральный процессор.
2. Выбранная команда расшифровывается и осуществляется переход к третьему шагу. Если же команда не входит в систему команд компьютера, то компьютер прекращает выполнение программы (ситуация АВОСТ).
3. Из оперативной памяти в процессор пересылаются операнды - данные, которые должны быть обработаны выполняемой командой.
4. Команда исполняется процессором и осуществляется переход к пятому шагу. Если же команду с выбранными данными выполнить не удалось, то возникает АВОСТ.
5. Результат выполнения команды пересылается в оперативную память.
6. Повторяется последовательность шагов, начиная с шага 1.

Программа аварийно завершается АВОСТом. Нормальное завершение программы осуществляется при исполнении команды СТОП.

Поскольку программа хранится в памяти, как и данные, она может быть модифицирована во время исполнения. Принцип хранимой в памяти компьютера программы впервые предложил американский ученый Джон фон Нейман. Он же предложил основные принципы построения универсальной вычислительной машины (архитектура фон Неймана).

Проанализировав последовательность шагов исполнения одной команды, можно заметить, что процессор трижды обращался к оперативной памяти (чтение команды, чтение операндов, запись результата). Отсюда ясно, что самым «узким» местом в компьютере является связь между процессором и оперативной памятью.

В современных компьютерах наряду с архитектурой фон Неймана используется магистральная архитектура. В ней все устройства компьютера подключаются к единой унифицированной линии связи - магистрали. Посылая по магистрали электрические сигналы, любой блок компьютера может передавать информацию

любому другому блоку или другому компьютеру по линии связи. Такая унификация правил и средств соединения устройств позволяет включать в состав компьютера различные новые устройства и заменять их при необходимости.

4. АЛГОРИТМ И ЕГО СВОЙСТВА

Понятие алгоритма относится к числу основных, самых общих понятий информатики, так же как понятия точки, линии, плоскости в геометрии. Поэтому определяется это понятие с помощью описания. Алгоритм - последовательность действий, ведущих от исходных данных к нужному результату, - удовлетворяет трем основным свойствам: 1) определенности (детерминированности); 2) массовости; 3) конечности.

Прежде всего это определение подчеркивает, что алгоритм носит дискретный характер (последовательность действий).

Не существует единого мнения по поводу того, что называть алгоритмом - процесс или его описание. Логично сохранить термин «алгоритм» за описанием процесса. Дело в том, что фактически единого процесса, соответствующего данному описанию, не существует. Каждой совокупности исходных данных соответствует свое протекание процесса, которое можно назвать реализацией алгоритма.

Свойство детерминированности состоит в понятности каждого действия алгоритма его исполнителю. Если это не так, то при исполнении алгоритма возникает АВОСТ, потому что исполнитель не может или не умеет исполнять непонятное действие.

Свойство массовости состоит в способности алгоритма решать не одну задачу, а целый класс подобных задач, отличающихся конкретными значениями исходных данных.

Свойство конечности гарантирует получение результата за конечное время. При этом не нужно прилагать интеллектуальных усилий, достаточно только точно и аккуратно выполнять все действия, указанные в алгоритме.

Например, алгоритмами являются правила выполнения арифметических операций в позиционной системе счисления, описание передвижения человека в городе от одной точки до другой, рецепт приготовления кулинарного блюда или лекарства, алгоритм деления отрезка прямой или угла пополам и т.д.

Алгоритмами не являются процесс рисования картины художником, процесс решения задачи, процесс воспитания человека и т.д..

Для решения задачи можно предложить несколько алгоритмов. Например, переместиться из одной точки города в другую можно несколькими способами.

Существуют задачи, для которых нельзя построить алгоритм. Эти задачи называются алгоритмически неразрешимыми. Например, такими задачами являются задача об удвоении куба (построить с помощью циркуля и линейки куб, объем которого в два раза больше объема данного куба), задача о разбиении угла на три равные части с помощью циркуля и линейки (трисекция угла) и т.п.

Алгоритмы используются при обработке данных на компьютере, при управлении космическими кораблями, для резервирования билетов на транспорт, для управления сложными объектами и т.п.

Пример 4.1. Составить алгоритм перехода перекрестка со светофором.

Решение. 1. Посмотреть на светофор. Если горит зеленый сигнал, то перейти к действию 3.

2. Стоять и ждать зеленого сигнала.

3. Переходить улицу до середины, смотря на транспорт, идущий слева.

4. Посмотреть на светофор. Если горит зеленый сигнал, то перейти к действию 6.

5. Стоять и ждать зеленого света.

6. Переходить улицу до конца, смотря на транспорт, идущий справа.

7. Конец.

Этот алгоритм рассчитан на человека, умеющего различать сигналы светофора, стоять и ждать, переходить и смотреть, умеющего определить середину улицы. Человек, не различающий или путающий цвета, не сможет выполнить уже первое действие этого алгоритма. Свойство массовости этого алгоритма состоит в том, что, руководствуясь им, можно перейти любой перекресток со светофором. Свойство конечности гарантирует, что руководствуясь этим алгоритмом, можно перейти улицу за конечное время при исправном светофоре. Если светофор не исправен, нужно воспользоваться другим алгоритмом.

Пример 4.2. Составить алгоритм варки картофеля.

Решение. 1. Взять кастрюлю такого объема, чтобы в нее вместился картофель, который требуется сварить.

2. Пока картофель не закончился, повторять:

   1) взять одну картофелину;

   2) вымыть ее;

   3) очистить от кожуры;

   4) положить вычищенную картофелину в кастрюлю.

3. Налить в кастрюлю воды так, чтобы она закрыла картофель.

4. Поставить кастрюлю на огонь.

5. Снять кастрюлю через 20 минут.

Этот алгоритм может выполнить исполнитель, понимающий и умеющий выполнять любую из команд, имеющихся в данном алгоритме. Если исполнитель не умеет мыть картофель, то произойдет АВОСТ и исполнение алгоритма не будет завершено. В этом заключается свойство определенности алгоритма.

Свойство массовости заключается в том, что, используя этот алгоритм, можно сварить различное количество картофеля: одну или две штуки, один или два килограмма и т.п.

Свойство конечности состоит в том, что вареный картофель будет получен после того, как исполнитель выполнит все шаги данного алгоритма.

Алгоритмы можно представить в разных формах. Различают словесное представление алгоритмов (в этом виде представлены алгоритмы в примерах 1 и 2), представление алгоритма в виде блок-схемы (рис. 4.1), представление алгоритма в виде программы на алгоритмическом языке. В дальнейшем для записи алгоритмов будем использовать алгоритмический язык Паскаль, разработанный швейцарским учёным Н. Виртом. Этот алгоритмический язык наиболее удобен для первоначального обучения программированию. Он позволяет выработать ясное алгоритмическое мышление, строить короткую и хорошо читаемую программу, продемонстрировать основные приёмы алгоритмизации.

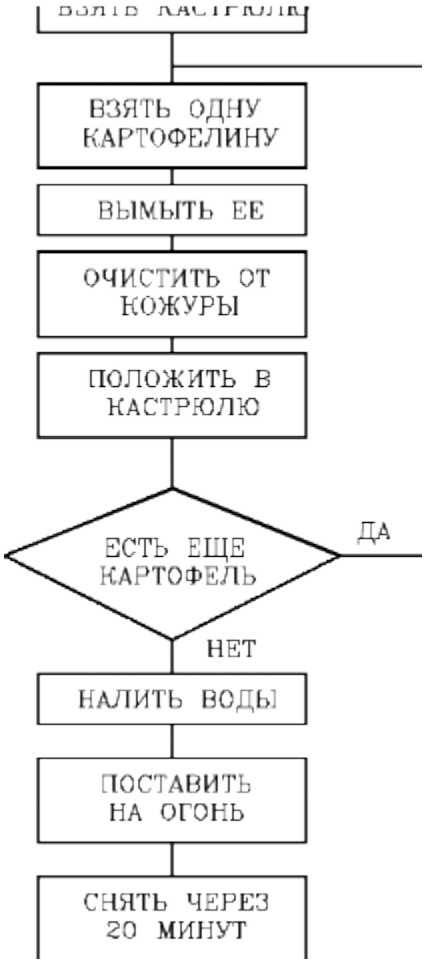


Рис. 4.1

5. Д А Н Н Ы Е

Данные отображают в программе окружающий мир. Отдельные представители окружающего мира имеют различные характеристики и объединяются в типы в соответствии с множеством операций, которые над ними можно исполнять. Например, зеленый свежий огурец весом 100 грамм относится к овощам, которые можно съесть свежими, засолить или порезать для салата и т.п. Характеристики объектов могут принимать различные значения из множества возможных значений, называемого доменом. Например, при описании человека значение его имени берется из домена, содержащего различные строки; дата рождения - из домена дат, а цвет глаз - из домена цветов.

Следует помнить, что значение характеристики обычно состоит из двух частей: количественной и качественной. Например, «его рост 173 см», «вес ящика яблок - 50 кг». Часто при обработке эти две части характеристики разрываются. Например, решая текстовую задачу на составление уравнения, мы можем получить квадратное уравнение. Решая его, можно получить два корня, например: целый и дробный. В ответе оставляем только целое значение, т.к. дробное не подходит, потому что по смыслу в ответе необходимо получить целое число.

Характеристика отдельного объекта может быть неизменной (не меняться в процессе вычислений). Такая характеристика представляется константой, т.е. явным изображением своего значения.

Для ссылок на изменяющиеся значения характеристики используются имена, называемые переменными. Имена строятся из букв и цифр, но начинаются они с буквы.

Если для задания характеристики достаточно одного значения, то ее называют скалярной (атомарной, неделимой).

Довольно часто значение характеристики имеет сложное строение. Такую характеристику называют структурной. Для ее задания обычно указывается несколько значений, связанных между собой определенным образом. Примеры таких характеристик: скорость, ускорение (задаются величиной и напрвлением), вектор на плоскости, состав некоторого класса школы, дата рождения и т.п.

Для каждого типа объекта можно выделить некоторое подмножество характеристик, которое однозначно определяет каждый отдельный объект внутри типа. Такое подмножество характеристик называется ключом. Например, для списка учеников некоторого класса ключом может быть порядковый номер ученика в журнале. Ключом для автомобиля может являться его регистрационный номер в ГАИ, а ключом для книги - ее автор и заглавие. Часто бывает, что несколько различных комбинаций характеристик могут быть использованы в качестве ключей. Какую из них выбрать в качестве ключа решает программист. Например, тип «семья» может иметь следующие характеристики: фамилию, адрес, телефон, наличие домашних животных или дачного участка. Кроме этих характеристик, для удобства обработки добавляется искусственная характеристика - порядковый номер семьи. Здесь в качестве ключа можно выбрать либо порядковый номер, либо фамилию при условии отсутствия однофамильцев, либо номер телефона при условии установки только одного телефона в каждой семье. Объекты разных типов взаимодействуют друг с другом. Чтобы описать эти взаимодействия обычно перечисляют те объекты, которые в него вступают. Списки таких объектов называют отношениями. Примерами отношений являются:

1. Списки семей, имеющих автомобили. В этом списке перечисляются номера семей и номера автомобилей, имеющихся у этих семей (предполагается наличие одного автомобиля у каждой семьи).

2. Списки учеников, получивших оценки по французскому языку за текущую неделю. В этом списке перечислены фамилии учеников и оценки, полученные ими по французскому языку.

3. Списки трансляторов, имеющихся на данном компьютере, где перечисляются алгоритмические языки и трансляторы для них.

Простейшими доменами, откуда черпаются значения данных, являются домены, содержащие целые и вещественные числа, логические значения и строковые данные. Такие домены представляют собой тип данных, т.к. содержат одинаковые элементы, над которыми разрешено выполнять только определенные для них операции. На основе этих (стандартных) доменов строятся более сложные домены.

5.1. Стандартный домен integer (целый тип)

Целый тип предназначен для представления в компьютере подмножества целых чисел. Значения данных этого типа представляются точно. Наибольшее целое число, представимое в Паскале, задается константой maxint. Чаще всего значение этой константы равно 32767.

Подмножество целых чисел, представимых в компьютере, упорядочено естественным образом:

-maxint < ... < -3 < -2 < -1 < 0 < 1 < 2 < 3 < ... < maxint.

Тип данных, для каждого значения которого (кроме первого и последнего) можно указать предыдущее и последующее значение, называется порядковым. Таким образом, целый тип относится к порядковым типам.

Для порядковых типов определены стандартные функции:

• succ(i) - возвращает следующее за i значение;

• pred(i) - возвращает предыдущее значение.

Целые константы записываются так же, как целые числа в математике: -132, +17, 349. Если знак у константы опущен, то константа считается положительной.

В Паскале имеется возможность присвоить имена константам. Для этого используется следующая конструкция: *const имя_константы=константа;*.

Пример 5.1:

```
const  dwa=2;          {константе 2 присвоено имя dwa}
       two=+2;         {константе 2 присвоено имя two}
       zwei=two;       {имени two присвоен синоним zwei}
       tri=dwa+1;      {имя tri присваивается константе, полученной как
                          значение выражения dwa+1}
       funf=zwei+tri;{имя funf присваивается константе, полученной как
                             значение выражения zwei+tri}.
```

Упражнение. Найдите и объясните ошибки в следующих описаниях констант:

```
CoNsT        dwa = 2;
       zwei = dwa + two;
       two = 4;
       funf = tri * two;
       tri = dwa + 1;
       two = funf div dwa; .
```

Целые переменные описываются следующим образом:

var имя :integer; или var имя1,имя2,...,имяN :integer; .

При трансляции этого оператора резервируется место в памяти для перечисленных переменных, устанавливается множество разрешенных над этими переменными операций.

Пример 5.2 описания переменных целого типа:

```
var i,j,k              :integer;     { здесь описаны три переменных }
    time,summa         :integer;     { имена переменных должны подсказывать
                                       их назначение }
    computer           :integer;     { здесь описана одна переменная }
```

Упражнение. Найдите и объясните ошибки в описаниях целых переменных:

```
var a,b,c          :integer;
    b,c,d,         :integer;
    const          :integer;
    dwa+tri        :integer;
    x,,y;          integer;
    p,r,s,t,u,f,v,w :integer;
```

Над целыми данными разрешены перечисленные ниже операции:

1) операции, формирующие целый результат:
   - ⇒+ сложение, например: 5+3=8;
   - ⇒- вычитание, например: 5-3=2;
   - ⇒* умножение, например: 5·3=15;
   - ⇒div деление до целых, например: 5 div 3 = 1;
   - ⇒mod остаток от деления до целых, например: 5 mod 3 = 2;

2) операции, формирующие вещественный результат:
   - ⇒/ деление, например: 5 / 3 = 1,67;

3) операции, возвращающие логический результат (true или false):
   - ⇒< меньше, например: 5<3 = false;
   - ⇒<= меньше или равно, например: 5<=5 = true;
   - ⇒> больше, например: 5>3 = true;
   - ⇒>= больше или равно, например: 5>=7 = false;
   - ⇒= равно, например: 8=8 = true;
   - ⇒<> неравно, например: 9<>9 = false.

Правила выполнения этих операций соответствуют правилам выполнения аналогичных операций в математике.

Для целого типа определены следующие стандартные (встроенные, т.е. написанные заранее профессионалами) функции:

- abs(i), возвращает абсолютную величину выражения i;

- sqr(i), возвращает квадрат выражения i;

- odd(i), возвращает логическое значение true (истина), если аргумент нечетное число. В случае четного числа возвращается false (ложь).

Упражнения:

1. Вычислите значения выражений:

```
 25 div 6      25 mod 6      25 div 5       25 mod 5
  3 div 5       3 mod 5      25 div 0       25.0 mod 5
 25 div 5.0    25 div (-6)   25 mod (-6)   -25 div 6
-25 div 6     -25 mod 6     -25 div (-6)   -25 mod (-6)
```

2. Определите операцию div через другие операции и стандартные функции.
3. Определите операцию mod через другие операции и стандартные функции.
4. Вычислите значения выражений:

```
succ(3)+pred(5),
pred(3)+succ(5),
succ(a)+pred(4),
succ(succ(b))+pred(pred(c)),
succ(pred(succ(6)))+pred(succ(pred(10))),
```

succ(-maxint),
pred(maxint).

5.2. Стандартный домен real (вещественный тип)

Вещественный тип (real) предназначен для моделирования множества вещественных чисел в компьютере. Из математики известно, что любое вещественное число можно записать в виде бесконечной десятичной дроби - периодической (для рационального числа) или непериодической (для иррационального числа). Бесконечную десятичную дробь в памяти компьютера представить не удается из-за ограниченности ее объема. Поэтому на каком-то знаке запись десятичной дроби обрывается. Следовательно, одна такая запись представляет бесконечное множество вещественных чисел, которые отличаются друг от друга цифрами в отброшенной части. Поэтому вещественные числа в компьютере представляются приближенно. Для этих чисел не устанавливается порядок, потому что из-за отброшенной части невозможно указать следующее и предыдущее число.

Вещественные константы записываются так же, как десятичные дроби в математике, только для отделения целой и дробной части используется точка вместо запятой. Такая форма записи называется записью с фиксированной точкой.

Пример 5.3. 3.14, +3.14, -3.14 .

Для научно-технических расчетов применяется и другая форма записи вещественных констант - экспоненциальная (с плавающей точкой). Константа, записанная в этой форме, имеет вид: *мантисса**E**порядок*.

Мантисса записывается либо как целая константа, либо как константа с фиксированной точкой, а порядок записывается как целое число. В качестве разделителя можно использовать заглавную или строчную латинскую букву e(E). Такая форма записи аналогична математической записи: мантисса$\cdot 10^{\text{порядок}}$. В этой форме одно значение можно представить разными записями, в которых изменяется положение десятичной точки и корректируется порядок. Отсюда другое название - форма представления вещественного числа с плавающей точкой.

Например, $0{,}5 \cdot 10^{-3}$ можно записать как 0.5E-3 или 0.5e-3, или 0.0005e0, или 5e-4 (отсюда название - с плавающей точкой).

Упражнение. Запишите константы Паскаля в традиционной записи:
-12.3E+2, -0.8e-6, 1E3, +1e-6.

Вещественные переменные описываются следующим образом:
var имя :real; или var имя1,имя2,...,имяN :real; .

Над данными вещественного типа выполняются следующие операции, формирующие вещественный результат:

=> + суммирование, например: 3,14 + 2,75 = 5,89;
=> - вычитание, например: 3,14 - 2,75 = 0,39;
=> * умножение, например: 1,2 · 6,3 = 7,56;
=> / деление, например: 7,2 / 9 = 0,8 .

Следующие операции при выполнении над данными вещественного типа возвращают логический результат: =, <, <=, >, >=, <> (неравно).

Стандартные функции, возвращающие вещественный результат при вещественном аргументе:

- abs(r), возвращает абсолютную величину r;

- sqr(r), возводит r в квадрат.

Стандартные функции, возвращающие вещественный результат при вещественном или целом аргументе:

- sin(r), возвращает синус r, аргумент задается в радианах;

- cos(r), возвращает косинус r, аргумент задается в радианах;

- arctan(r), возвращает арктангенс r, аргумент задается в радианах;

- ln(r), возвращает натуральный логарифм r;

- exp(r), возвращает экспоненту r;

- sqrt(r), возвращает квадратный корень из r.

Стандартные функции, возвращающие целый результат при вещественном аргументе:

- trunc(r), возвращает целую часть r. Дробная часть отбрасывается. Следовательно, trunc(3.7)=3, а trunc(-3.7)=-3.

- round(r), возвращает округленное до целых r. При r>=0 round(r) означает trunc(r+0.5), при x<0 - trunc(x-0.5).

Пример 5.4. Вычислите a в степени в для положительного a.

Решение. Exp(b*ln(a)).

Пример 5.5. Вычислите tg(α).

Решение. sin(alpha)/cos(alpha).

Упражнения: 1. Запишите выражения на Паскале:

1) x^5 ; 2) $\cos^7 x^4$; 3) $\log_6 x/5$;

4) $|x^{-3}|$; 5) arcsin x ; 6) sin 8° .

2. Вычислите значения выражений и сравните результаты разных функций для одинаковых аргументов:

trunc(2.8), round(2.8), trunc(2.2), round(2.2),
trunc(-1.8), round(-1.8), round(0.5), round(-0.5).

5.3. Стандартный домен char (символьный тип)

Значениями символьного типа являются элементы конечного и упорядоченного множества символов. В каждом компьютере это множество может быть разным, но чаще всего оно определяется множеством значений кодовой таблицы ASCII.

Для прямого и обратного отображения множества символов в подмножество натуральных чисел - порядковых номеров для множества символов - существуют две стандартные функции:

- (c), возвращает порядковый номер символа c в кодовой таблице ASCII (целое число);

- (i), возвращает символ с порядковым номером i.

Функции ord и chr обратные по отношению друг к другу, т.е. chr(ord(c))=c и ord(chr(i))=i.

Пример 5.6. число = chr('цифра')-ord('0'),
9 = chr('9') -ord('0'),
5 = chr('5') -ord('0').

Этот тип является порядковым. Значения его упорядочены так, что c1<c2, если ord(c1)<ord(c2). Это определение распространяется на любую из операций типа сравнения: =, <, <=, >, >=, <> (неравно).

Для аргументов типа char стандартные функции pred и succ определяются так: pred(c) = chr(ord(c)-1), succ(c) = chr(ord(c)+1).

5.4. Стандартный домен string (строковый тип)

Значениями этого типа данных являются последовательности символов разной длины. Здесь могут использоваться любые символы, имеющиеся в кодовой таблице компьютера. Каждая строка характеризуется своей текущей длиной (количеством символов, имеющихся в строке в данный момент), порядком расположения символов. По умолчанию для строк определена максимальная длина, равная 255 символам, но ее можно изменить, указав при описании: *string[число]*. Если строка длиннее максимальной длины, то не поместившиеся символы отбрасываются.

Из изложенного следует, что данный тип не является скалярным, т.к. каждое данное определяется двумя величинами: длиной и порядком расположения символов внутри константы. К тому же последовательность символов имеет не простое строение (структуру), т.к. для каждого символа в строке можно указать его соседей слева и справа (если они есть). Значения этого типа упорядочены в лексикографическом порядке в соответствии с кодовой таблицей так, что меньшая строка предшествует большей.

Представление строки в памяти компьютера может быть осуществлено двумя способами. В первом способе строке предшествует элемент, в котором хранится длина строки. Например,

5	А	Л	И	С	А

'АЛИСА'

или

0			

или " (пустая строка).

Во втором способе строку завершает специальный признак окончания строки. Например,

А	Л	И	С	А	\0

'АЛИСА'

или

\0			

или " (пустая строка).

Строковые константы записываются в апострофах. Например:

- 'текст', длина этой константы 5 символов;
- 'Азбука Морзе', длина этой константы 12 символов;
- '', длина этой константы 0 символов (пустая строка).

Строковые переменные описываются в операторе var с описателем string:

var список_переменных : string;

или *var список_переменных :string[максимальная_длина_число];*.

Например, пусть описана переменная *var a: string[3];* Тогда при присваивании переменной а разных констант можно получить следующее содержимое памяти компьютера:

a:='';

0			

a:='У';

1	У		

a:='УА';	2	У	А	
a:='КОТ';	3	К	О	Т
a:='КОШКА';	3	К	О	Ш

Над строками выполняется операция сцепления (конкатенации), которая позволяет соединить две или более строк в одну без разделителей. Например, 'Кро'+'код'+'ил' позволит получить новую строку 'Крокодил'.

Над строками выполняются операции сравнения: =, <, <=, >, >=, <>. Строки сравниваются посимвольно слева направо до первого результата или до исчерпания символов строки. Символы сравниваются как данные типа char.

Например, 'школа'<'школьник' Результат сравнения «истина» (true), т.к. 'ш'='ш', 'к'='к', 'о'='о', 'л'='л', 'а'<'ь'(символ 'а' расположен в кодовой таблице раньше символа 'ь').

'АЗБУКА'='АЗБУКА', т.к. все символы поэлементно совпадают. Кроме операций, для строк имеются стандартные функции и процедуры.

5.4.1. Функции

СЦЕПЛЕНИЕ - CONCAT(строка1,строка2,...). Аналогична операции сцепления.

Пример 5.7. Исходные данные: a = 'код', b = 'ил'.

Оператор: s := concat('кро',a,b).

Результат: S = 'крокодил'.

КОПИРОВАТЬ - COPY(строка,число1,число2). Из указанной строки выделяется подстрока, начиная с позиции, заданной числом 1, длиной, заданной числом 2.

Пример 5.8. Исходные данные: S='крокодил'.

Оператор: b:=copy(S,2,3).

Результат: b = 'рок'.

ПОЗИЦИЯ - POS(строка1,строка2). Отыскивает первое вхождение строки 1 в строке 2 и возвращает номер начальной позиции вхождения или ноль, если строка 1 не входит в строку 2.

Пример 5.9. Исходные данные: S = 'крокодил'.

Оператор: i := pos('око',S).

Результат: i = 3.

Оператор: i := pos('я','крокодил').

Результат: i = 0.

ДЛИНА - LENGTH(строка). Возвращает длину строки - аргумента.

Пример 5.10. Исходные данные: S = 'крокодил'.

Оператор: j := length(S).

Результат: j = 8.

5.4.2. Процедуры

ВСТАВИТЬ - INSERT(строка1,строка2,число). Вставляет строку 1 в строку 2, начиная с позиции, заданной числом. Если в результате получается строка длины больше максимальной, то она усекается справа.

Пример 5.11. Исходные данные: S = 'крокодил'.

Оператор: d := copy(S,3,3).

Результат: d = 'око'.

Оператор: insert('Н', d, 3).

Результат: d = 'окно'.

УДАЛИТЬ - DELETE(строка,число1,число2). Удаляет из строки подстроку, начиная с позиции, заданной числом 1, длиной, заданной числом 2. Если число 1 больше размера строки, то подстрока не удаляется. Если число 2 больше имевшегося количества, то удаляются символы до конца строки.

Пример 5.12. Исходные данные: S = 'крокодил'.

Оператор: delete(S,4,3).

Результат: = 'кроил'.

Оператор: delete(S, 1, 1).

Результат: S = 'роил'.

ПРЕОБРАЗОВАТЬ число в строку - STR(число[:M[:N]],строка). Преобразует число в строку. M задает общее количество символов, получаемых в строке, N - для вещественных чисел (типа real) задает количество цифр в дробной части.

Пример 5.13. str(123,S).

Результат: S = '123'.

ПРЕОБРАЗОВАТЬ строку в число - VAL(строка,число,код). Преобразует строку символов во внутреннее представление числа. Код указывает номер неправильного символа или равен 0 в случае успешного преобразования. ***Примеры 5.14***. val('+12.3',V,K).

Результат: V = 12.3, K = 0 {преобразование прошло успешно}.

val('23+5',v,k).

Результат: v = неопределено. k = 3 {ошибка при попытке преобразовать третий символ}.

Пример 5.15. Проверить, является ли заданная строка S строчной гласной буквой русского алфавита?

Решение. pos(S,'аэоуыяеёюи')>0.

Пример 5.16. Проверить, является ли заданная строка S строчной согласной буквой русского алфавита?

Решение. pos(S,'аэоуыяеёюи')=0.

Пример 5.17. Выделить часть строки после первого пробела.

Решение. copy(s,pos(' ',s)+1,length(s)-pos(' ',s))

Пример 5.18. Удалить последний символ строки S.

Решение. delete(S,length(S),1).

Пример 5.19. Записать оператор, позволяющий получить из слова «моряк» слово «морячок».

Решение. Пусть s='моряк', тогда insert('чо',s,5).

Пример 5.20. Выделить из строки S подстроку между i-й и j-й позициями, включая эти позиции.

Решение. copy(s,i,j-i+1).

Упражнения: 1. Пусть объявлена переменная var a:string[11]. Какое значение будет содержать эта переменная после присваивания a:='сумей'+', догадайся'.

2. Подставьте вместо знаков вопроса буквы русского алфавита так, чтобы получить осмысленное слово и не нарушить истинность логического выражения:

а) 'конфета' < 'конф??' = true,

б) 'ко????' > 'конфета' = false.

3. Из слова 'понедельник', используя функции строковых данных, получите слово 'плодик'.

4. Какие из следующих строк можно преобразовать в числа:

а) '1e+1' ,

б) ' -1.256' ,

в) '+295.689e6 ' ?

5.5. Стандартный домен Boolean (логический тип)

Логические данные принимают истинностные значения: «ложь» или «истина». Константы этого типа обозначаются на Паскале false и true. Этот тип относится к порядковым, константы упорядочены так: false<true.

Переменные логического типа описываются следующим образом:

var список_переменных :boolean; .

Данные этого типа называются булевскими в честь Джорджа Буля, разработавшего алгебру логики.

Упражнение. Дочь Буля - Этель Лилиан Войнич - автор романа «Овод». Известны ли вам еще родственники, один из которых сыграл важную роль в информатике, а другой - в литературе?

Для данных этого типа определены операции not (инверсия, логическое не, отрицание), and (конъюнкция, логическое и), or (дизъюнкция, логическое или). Эти операции задаются с помощью таблиц истинности.

Одноместная логическая операция отрицание заключается в употреблении перед логическими данными частицы «не» или слов «неверно, что». Обозначается операция not A, где A - логическое данное.

Таблица истинности для отрицания:

A	**not A**
true	false
false	true

Как видно из таблицы, эта операция меняет значение истинности на противоположное.

Примеры 5.2:

Логическое данное	**Его отрицание**
a>b	a<=b
a=b	a<>b
a<b	a>=b
a - четное	a - нечетное

На Земле существует вид животных, неизвестный человеку	Человеку известны все виды животных, обитающих на Земле.
Во всякий треугольник можно вписать окружность	Неверно, что во всякий треугольник можно вписать окружность или (другой вариант) существует треугольник, в который нельзя вписать окружность

Двуместная логическая операция - конъюнкция - вырабатывает значение «истинно» в случае, если оба логических данных, участвующих в этой операции, истинны. В остальных случаях вырабатывается значение «ложь». Эта операция обозначается в Паскале and и задается следующей таблицей истинности:

A	**B**	**A and B**
false	false	false
false	true	false
true	false	false
true	true	true

Примеры 5.22:

A	**B**	**A and B**
Светит солнце	Ветер не дует	Светит солнце И ветер не дует
a>0	a<5	(a>0) and (a<5) иначе 0<a<5
a=3	a<>6	(a=3) and (a<>6) иначе a=3
a<=1	a>=-1	(a<=1) and (a>=-1), иначе -1<=a<=1, иначе abs(a)<=1

Двуместная логическая операция - дизъюнкция - вырабатывает значение «ложно» в случае, если оба логических данных, участвующих в этой операции, ложны. В остальных случаях вырабатывается значение «истина». Эта операция обозначается в Паскале or и задается следующей таблицей истинности:

A	**B**	**A or B**
false	false	false
false	true	true
true	false	true
true	true	true

Примеры 5.23:

A	**B**	**A or B**
Тепло	Нет дождя	Тепло или нет дождя
a<6	a=6	(a<6) or (a=6) иначе a<=6

5.5.1. Законы логических операций

Коммутативные (переместительные) законы:

A and B = B and A, A or B = B or A.

Например, фразы «Дует ветер и идет дождь» и «Идет дождь и дует ветер» эквивалентны. Фразы «Тепло или нет дождя» и «Нет дождя или тепло» также эквивалентны. Однако в естественном языке эти законы, в отличие от логики и информатики, выполняются не всегда. Сравните, например, фразы «Он получил выговор и подрался» и «Он подрался и получил выговор».

Ассоциативные (сочетательные) законы:

A and (B and C) = (A and B) and C,

A or (B or C) = (A or B) or C.

Законы показывают, что если несколько одинаковых операций стоят рядом, то их можно выполнять в произвольном порядке, результат от этого не изменится.

Дистрибутивные (распределительные) законы:

A and (B or C) = (A and B) or (A and C),

A or (B and C) = (A or B) and (A or C).

Фразы «Пошел в школу и получил пять или три» и «Пошел в школу и получил пять или пошел в школу и получил три» эквивалентны. В арифметике для операций умножение и сложение справедлив только один дистрибутивный закон: a(b+c)=ab+ac.

Законы де Моргана:

not(A and B) = not A or not B,

not(A or B) = not A and not B.

Найдем отрицание фразы «Солнце светит и греет». Возможны два варианта:

1. Неверно, что солнце светит и греет (левая часть закона).
2. Солнце не светит или не греет (правая часть закона).

Найдем отрицание фразы «Речка движется или не движется». Возможны два варианта:

1. Неверно, что речка движется или не движется (левая часть закона).
2. Речка не движется и движется. (правая часть закона).

Упражнение. Огастес де Морган известен своими работами по математической логике, истории и математическим играм. Ему было, по собственному выражению, «x лет в году x^2», а умер он в возрасте 64 года в прошлом столетии (19-м веке). В каком году родился де Морган?

Законы *идемпотентности* (законы отсутствия степеней и коэффициентов):

A and A = A, A or A = A.

Законы *поглощения*:

A and (A or B) = A, A or (A and B) = A.

Закон *склеивания*: (A and B) or (A and not B) = A.

Эти законы используются для упрощения логических выражений.

Закон *исключенного третьего*:

A or not A = true.

Быть или не быть - третьего не дано.

Закон *противоречия*:

A and not A = false.

Не могут быть одновременно истинными высказывание и его отрицание.

Закон *двойного отрицания*:

not not A = A.

«Не верно, что я не пойду сегодня в кино» эквивалентно «Я пойду сегодня в кино».

Законы, следующие из таблиц истинности, определяющих операции:

A and false = false, A or false = A,

A and true = A, A or true = true,

not false = true, not true = false.

Для проверки справедливости логических законов используют таблицы истинности. В таблице истинности для всех возможных наборов исходных данных вычисляются значения левой и правой частей. Вычисленные значения сравниваются. Для справедливости закона необходимо совпадение значений обоих столбцов для всех возможных значений исходных данных. Такой подход возможен в алгебре логики благодаря небольшому количеству исходных данных. В качестве примера проверим один из дистрибутивных законов: A or (B and C) = (A or B) and (A or C).

В этот закон входят три логические переменные, каждая из которых может принимать одно из двух значений: «истина» или «ложь», следовательно, возможно 2^3=8 разных комбинаций этих логических переменных, поэтому таблица истинности будет содержать восемь строк:

A	**B**	**C**	**B and C**	**A or (B and C)**
false	false	false	false	false
false	false	true	false	false
false	true	false	false	false
false	true	true	true	true
true	false	false	false	true
true	false	true	false	true
true	true	false	false	true
true	true	true	true	true

В начале вычисляем результат операции B and C. Затем определяем результат дизъюнкции переменной A и столбца B and C. Этой операцией завершается вычисление левой части закона.

Приступаем к проверке правой части закона. Вычислим сначала результат операции A or B, затем - A or C. Наконец, найдем результат конъюнкции двух вновь полученных столбцов, завершая этим вычисление правой части закона:

A	**B**	**C**	**A or B**	**A or C**	**(A or B) and (A or C)**
false	false	false	false	false	false
false	false	true	false	true	false
false	true	false	true	false	false
false	true	true	true	true	true
true	false	false	true	true	true
true	false	true	true	true	true

true	true	false	true	true	true
true	true	true	true	true	true

Сравнив левую и правую части, убеждаемся, что они совпадают на всех возможных наборах исходных данных:

A or (B and C) левая часть	**совпадение**	**(A or B) and (A or C) правая часть**
false	совпадает	false
false	совпадает	false
false	совпадает	false
true	совпадает	true
true	совпадает	true
true	совпадает	true
true	совпадает	true
true	совпадает	true

Упражнение. Докажите с использованием таблиц истинности все законы логики.

6. ВЫРАЖЕНИЯ

Выражение - это константа, переменная, стандартная функция или их комбинация со знаками операций и круглыми скобками.

Выражения записывают в строчку, надстрочные и подстрочные индексы запрещены. Переносить выражение можно по знаку операции, причем сам знак операции в новой строке не повторяется.

В выражении два знака операций рядом не ставятся. Для их разделения используются скобки. При записи выражения знак умножения опускать нельзя, иначе может быть получено новое имя и смысл выражения изменится или может быть получена бессмыслица, начинающаяся с цифры. Например, 2*q может превратиться в 2q, а q*2 - в q2.

Все данные в выражении должны быть одного типа, причем значение выражения получается того же типа. Исключение допускается для данных целого типа, которые без ограничений могут использоваться в выражениях с вещественными данными, причем результат получается вещественным. Результатом выражений типа сравнения всегда является логическое данное.

Рассмотрите примеры записи выражений. Слева приводится запись выражения по правилам математики, справа - по правилам Паскаля:

$a + bx + cyz$	a+b*x+c*y*z
$\frac{ab}{c} + \frac{c}{ab}$	a*b/c+c/(a*b)
$\frac{x + y}{a_1} \cdot \frac{a_2}{x - y}$	(x+y)/a[1]*a[2]/(x+y)
$10^4\alpha - 3\frac{1}{5}\beta$	1e4*alpha-3.2*beta
$ax^2 + bx + c$	a*sqr(x)+b*x+c

$\sqrt{1+\frac{x^2}{\lvert a+bx\rvert}}$	sqrt(1+sqr(x)/abs(a+b*x))
$a+\cfrac{b}{c+\cfrac{d}{e+\cfrac{f}{gh}}}$	a+b/(c+d/(e+f/(g*h)))
$\log_2 \frac{x}{5}$	ln(x/5)/ln(2)
$\sin x^2$	sin(sqr(x))
$\cos^2 x$	sqr(cos(x))
a^b	exp(b*ln(a))

Значение выражения вычисляется в следующем порядке. Сначала вычисляются части выражения, заключенные в круглые скобки, и функции. Затем учитываются приоритеты операций. В Паскале приняты следующие приоритеты:

1. not, - (унарный), + (унарный).
2. *, /, div, mod, and.
3. +, -, or.
4. <, <=, >, >=, =, <>.

Если несколько операций одного приоритета расположены рядом, то они выполняются в порядке записи слева направо.

Упражнения:

1. Укажите порядок вычисления значения выражений:
   а) a or b and not a,
   б) (x>=0) or t and odd(x) or (sqr(y)<>4),
   в) a-b-c-d,
   г) -a mod b+a div b*c,
   д) 3+7 div 2 mod 7/2-trunc(sin(abs(-1))).
2. Преобразуйте выражение 1в так, чтобы операции выполнялись в обратном порядке.

7. СТРУКТУРА ПРОГРАММЫ НА АЛГОРИТМИЧЕСКОМ ЯЗЫКЕ ПАСКАЛЬ

Константы, переменные, выражения, специальные символы используются для записи программы. Программа соответствует алгоритму. Эта форма представления алгоритма отличается точностью и однозначностью каждой фразы. Фраза на Паскале называется оператором. Операторы предназначены для описания данных, используемых в программе, или для указания действий для обработки данных. Ниже приводится структура Паскаль-программы:

```
rogram имя_программы;                    { 1 }
писание констант;                        { 2 }
писание типов;                           { 3 }
писание переменных;                      { 4 }
писание процедур и/или функций;          { 5 }
begin                                    { 6 }
```

```
    операторы, разделяемые точкой с запятой      { 7 }
end.                                              { 8 }
```

В любом месте Паскаль-программы там, где можно поставить пробел или другой разделитель, можно поставить комментарий. Комментарий - это последовательность любых символов, заключенная в фигурные скобки. Внутри комментария другие комментарии не разрешены. Комментарий используется для пояснения текста программы и компьютером не обрабатывается. Он нужен для человека, работающего с программой, чтобы облегчить ее понимание. Хорошие комментарии должны содержать некоторую дополнительную информацию, а не перефразировать программу. Например, если i - счетчик количества ящиков с яблоками, то второй комментарий предпочтительнее.

{ 1 } i:=i+1; { i увеличить на единицу },

{ 2 } i:=i+1; { количество ящиков с яблоками увеличиваем на 1 }.

Комментарии нужно располагать так, чтобы они не делали программу менее наглядной. Сравните два приведенных ниже фрагмента и убедитесь в этом.

{ 1 } a:=b+{ это сложение b и c }c; ,

{ 2 } a:=b+c; { b-проданное, c-оставшееся количество яблок }.

Неправильные комментарии хуже, чем их отсутствие, поскольку такие комментарии вводят в заблуждение. Поэтому комментарии должны изменяться вместе с изменениями программы. Пример неправильного комментария приведен ниже:

```
if i mod 2=1                  { если i четное }
then write(i,' - нечетное')   { напечатать }
else write(i,' - четное')     { тоже напечатать }
```

Упражнение. Когда будет изучено несколько операторов Паскаля, напишите программу без комментариев. Сохраните ее и вернитесь к ней через две-три недели. Постарайтесь вспомнить и разобраться в написанной ранее программе. Сравните затраченные при разборе программы усилия с усилиями, потраченными на первоначальное ее создание. Прокомментируйте мнение программистов: «Лучше разработать программу заново, чем разбираться в чужой, не комментированной программе».

Комментарием { 1 } в структуре Паскаль-программы отмечен заголовок. Он начинается с символа program (для компьютера эти семь латинских букв - один символ). Затем указывается имя программы, строящееся, как и любое имя, из латинских букв, цифр и знака подчеркивания. Начинается имя всегда с буквы.

После заголовка программы рекомендуется вставлять комментарий, содержащий постановку задачи, обозначения исходных данных и результатов, используемые алгоритмы и функции, краткое указание на идею решения задачи, реализованную в программе.

Описание констант (отмечено комментарием { 2 }) предназначено для присвоения некоторым константам имен. Имена в тексте программы заменяют конкретное значение константы. Это позволяет изменять значение константы при различных исполнениях программы без просмотра и изменения всей программы. Здесь имеется аналогия со следующей, распространенной в реальной жизни, ситуацией, когда, например, заранее печатаются бланки договора с детальным определением обязанностей «заказчика» и «исполнителя», а в самом начале

договора имеются строки вида «фамилия, имя, отчество, именуемый в дальнейшем «заказчик», и фамилия, имя, отчество, именуемый в дальнейшем «исполнитель». Эти строки устанавливают назначения конкретных людей для исполнения заранее описанных функций. Таким образом, одна и та же форма может использоваться для заключения договора между разными людьми. Описанным константам отводится место в оперативной памяти компьютера.

Описание типов (отмечено комментарием { 3 }) предназначено для определения типов пользователя. Тип определяет множество возможных значений данного и множество разрешенных над данными этого типа операций.

Описание переменных (отмечено комментарием { 4 }) предназначено для резервирования места в оперативной памяти используемым переменным. Реально место в памяти отводится для описанных переменных в момент передачи управления программе (вызова ее для исполнения). По окончании работы программы она удаляется из оперативной памяти вместе со всеми переменными, константами, типами, вложенными процедурами и функциями. Эта операция освобождает память для других программ и данных.

Для каждой переменной указывается ее тип. Это позволяет уже на этапе трансляции программы выявить некоторые ошибки, например, ошибки в имени (tyme вместо time), запрещенные операции (a/b для данных string), выход за границу допустимых значений (i:=2*k для данных из диапазона 1..k).

Описание процедур и/или функций (отмечено комментарием { 5 }) предназначено для описания вспомогательных алгоритмов, которые применяются для облегчения программирования, поскольку позволяют использовать в данной разработке алгоритмы, построенные ранее. Описанные процедуры называются вложенными и существуют (могут быть выполнены) во время исполнения программы.

После описаний следует блок действий. Он начинается с символа begin (комментарий { 6 }). Завершается символом end, за которым обязательно следует точка (комментарий { 8 }).

Внутри блока записываются операторы (комментарий { 7 }), разделяемые символом - точка с запятой. Обратите внимание, что операторы разделяются точкой с запятой. Из этого следует, что если после некоторого оператора следует другой оператор, то последний отделяется от предыдущего точкой с запятой. Таким образом, количество операторов в блоке равно количеству точек с запятой плюс один. Программисты часто ошибались, ставя точку с запятой и после последнего оператора. Для избежания большого количества ошибок такую ситуацию транслятор Паскаля рассматривает как употребление пустого оператора. Пустой оператор ничего не делает, а используется для удобства записи и уменьшения количества ошибок. Например, в записи i:=i+2; имеется два оператора. Один - оператор присваивания, а другой - пустой.

8. ЛИНЕЙНЫЕ АЛГОРИТМЫ

Простейшим способом построения новых алгоритмов является последовательное исполнение команд исполнителя и других алгоритмов, написанных для данного исполнителя ранее. Такие алгоритмы называются линейными. Запись команд для исполнителя осуществляется на алгоритмических

языках операторами. Для записи линейных алгоритмов используются операторы ввода, вывода, присваивания, оператор процедуры.

8.1. Оператор ввода

Он предназначен для приема (получения) от человека исходных данных и размещения их в памяти компьютера. Форма записи этого оператора в Паскале может быть одной из трех:

- *read(список скалярных переменных) ,*
- *readln(список скалярных переменных) ,*
- *readln .*

При выполнении этого оператора компьютер остановится и будет ждать, когда пользователь (человек, работающий за компьютером) введет столько констант, сколько переменных указано в списке оператора. Вводимые константы разделяются одним или несколькими пробелами и заносятся в специальный буфер - буфер клавиатуры. Признаком окончания ввода является нажатие клавиши Enter. Код этой клавиши, как и введенные константы, заносится в буфер клавиатуры. После нажатия клавиши Enter компьютер продолжит исполнение оператора ввода. Первая константа пересылается в первую переменную списка, вторая - во вторую и т.д. Если пользователь ввел меньше констант, чем переменных в списке, то ожидание окончания ввода будет продолжаться.

По окончании ввода в буфере клавиатуры останется признак нажатия клавиши Enter. Чтобы его убрать из буфера, в операторе используется суффикс ln.

Оператор ввода readln заставит компьютер ожидать до тех пор, пока в буфере клавиатуры не появится признак нажатия клавиши Enter.

Пример 8.1. Какие значения получат переменные a, b, c после выполнения оператора ввода, если на клавиатуре набраны константы
3.14 -3.14 +6.28 Enter?

```
var a,b,c:integer;
begin read(a,b,c)
end.
```

Решение. После выполнения указанного оператора получим a=3.14, b=-3.14, c=6.28. В буфере клавиатуры останется признак нажатия клавиши Enter.

Пример 8.2. Тот же вопрос для набранных констант 3 5 -7.2 Enter.

```
Var     a,b:real;
        c   :integer;
begin readln(a,c,b)
end.
```

Решение. После выполнения оператора получим a=3.0, b=-7.2, c=5. Буфер клавиатуры будет пуст.

Пример 8.3. Тот же вопрос для набранных констант 1 2 3 Enter.

```
var a,b:integer;
begin readln(a,b,a)
end.
```

Решение. После выполнения оператора получим a=3, b=2. Буфер клавиатуры будет пуст.

8.2. Оператор вывода

Оператор вывода предназначен для передачи данных из оперативной памяти компьютера и преобразования их в форму, удобную для восприятия человеком. Его можно записать в одной из трех форм:

- *write(список выражений) ,*
- *writeln(список выражений) ,*
- *writeln .*

Если сравнивать этот оператор с оператором ввода, то можно заметить много общего в их устройстве. Прежде всего и тот, и другой операторы существуют в трех формах, начинаются со служебного символа, который может иметь суффикс ln, далее может следовать список объектов, в котором объекты перечисляются через запятую. В операторе ввода объектами могут быть только переменные, тогда как в операторе вывода объектом может быть выражение, что является общим случаем по сравнению с переменной.

При выполнении оператора вывода сначала вычисляются значения выражений из списка, а затем выражения выводятся в порядке их следования в списке на экран с того места, где находился курсор. По окончании вывода курсор остается в следующей свободной позиции. Если в записи оператора присутствует суффикс ln, то по окончании вывода курсор автоматически переводится в первую позицию следующей строки.

Для выражений в списке оператора вывода можно указать размерность вывода. Для данных целого типа размерность задается в виде: *данное:число*, где число указывает количество позиций, которое должно занять значение на экране. Если количество позиций окажется меньше, чем нужно для вывода, то компьютер автоматически добавит недостающие позиции. Для данных типа real размерность задается в виде: *данное:число1:число2*, где число 1 задает общее количество позиций для представления значения данного, а число 2 указывает количество позиций в дробной части. Если ширина поля не указана, то она определяется компьютером.

Пример 8.4. Что будет выведено в результате выполнения оператора write('x1=',2,' x2=',3*a), если a=2?

Решение. Будет выведена строка x1=2 x2=6.

Пример 8.5. Какие строки будут напечатаны последовательностью операторов вывода: write(1); write(2,3); writeln(4); writeln(5,6); writeln;writeln(7,8)?

Решение. Ниже приводится вид строк с указанием операторов, с помощью которых получена каждая строка. По окончании выполнения операторов курсор будет переведен в новую (пятую по счету) строку:

строка 1:1234	write(1);write(2,3);writeln(4);
строка 2:56	writeln(5,6);
строка 3:	writeln;
строка 4:78	writeln(7,8)
строка 5:_	

8.3. Оператор присваивания

Предназначен для изменения содержимого оперативной памяти. Старое значение при этом стирается и записывается новое. На Паскале этот оператор записывается так: *переменная:=выражение*.

Исполняется оператор в два этапа: 1) вычисляется значение выражения, записанное справа; 2) вычисленное значение пересылается в переменную слева.

Примеры:
```
a:=5;
s:='оператор присваивания';
t:=sqr(x)-sqr(y);
i:=i+1;
```

8.4.Примеры построения линейных алгоритмов

Пример 8.6. Заданы два числа, которые хранятся в переменных a и b. Необходимо поменять местами значения этих переменных.

Решение. Итак, заданы две переменные, которые хранят числовую информацию. Числовая информация в алгоритмическом языке может быть представлена как целая или вещественная (приближенная). Поскольку в условии задачи нет явного указания на тип числовой информации, будем использовать более общий тип - вещественный. Исходными данными могут быть, например, такие числа:

Исходные данные		**Результат**	
a	**b**	**a**	**b**
5	3	3	5
3.14	2.81	2.81	3.14
-40	+15	+15	-40

Попытаемся решить задачу с помощью оператора присваивания a:=b. После его выполнения значения переменных a и b будут одинаковыми, но будет утеряно первоначальное значение переменной a. Для того, чтобы его сохранить, поместим его в рабочую переменную r. Тогда последовательность операторов, решающая задачу, будет такой:

```
r:=a;{ запомнили значение переменной a }
a:=b;{ в переменную a записали значение переменной b }
b:=r { в переменную b записали сохраненное в r старое значение a }
```

После выполнения этих операторов значения переменных поменялись местами. Решение получено, но, помня о том, что для решения задачи можно, как правило, предложить несколько алгоритмов, продолжим поиски. Постараемся найти алгоритм, который для решения задачи не требует введения вспомогательной переменной. Заметим, что исходные данные представляют собой числа. Над числами можно выполнять арифметические операции. Без потери значений можно найти сумму, пусть a:=a+b. Теперь нам известна сумма заданных значений и исходное значение переменной b. С помощью вычитания можно найти значение a и поместить в b, т.е. b:=a-b. Теперь нам известна сумма заданных значений и

исходное значение переменной a, которое хранится в переменной b. Чтобы решить задачу, осталось выполнить a:=a-b.

Проведем трассировку найденных операторов.

a:=a+b;

b:=a-b;

a:=a-b

Пусть первоначально a=5, b=3. После исполнения оператора a:=a+b получим a=8, b=3. После исполнения оператора b:=a-b значения переменных изменятся a=8, b=5. После исполнения оператора a:=a-b окончательно получим a=3, b=5.

Можно получить еще одно решение, если воспользоваться двоичным представлением чисел. Над двоичными числами можно выполнять операцию исключающего ИЛИ(xor), которая задается следующей таблицей:

x	**y**	**x xor y**
0	0	0
0	1	1
1	0	1
1	1	0

С использованием двоичного представления чисел и операции xor поставленную задачу решает приведенная ниже последовательность операторов.

Пусть a=5 в двоичном представлении 0101, b=11 в двоичном представлении 1011.

```
a:= a xor b    xor 0101
                   1011
                  ----
                   1110
```

Результат помещаем в a.

```
b:= a xor b    xor 1110
                   1011
                  ----
                   0101
```

Результат помещаем в b.

```
a:= a xor b    xor 1110
                   0101
                  ----
                   1011
```

Результат помещаем в a.

Окончательно получаем a=1011, b=0101.

Из трех рассмотренных алгоритмов два последних не используют дополнительной переменной, причем в последнем - только одна операция. Однако последний алгоритм можно применить к двоичным данным (любым, хранящимся в виде двоичных последовательностей), второй - только к числовым. Первый алгоритм имеет универсальный характер: с его помощью можно поменять местами не только числа, но и различные предметы: мел и тряпку, пару бутылок или людей, записи на магнитофонных кассетах и т.д. К тому же этот алгоритм представляется наиболее наглядным. Приведем запись этого алгоритма на Паскале:

```
program obmen;
{ поменять местами значения двух переменных }
```

```
var a,b:real; { исходные данные и результат }
    r:real; { вспомогательная переменная }
begin  write('Введите два числа ');
       readln(a,b);
       r:=a; { 1 }
       a:=b; { 2 }
       b:=r; { 3 }
       writeln('результат ',a,' ',b)
end.
```

Остановимся еще на некоторых выводах, которые можно сделать из решения задачи 1. Решение начинается с выбора способа представления исходных данных. Этот выбор во многом определяет алгоритм решения задачи, потому что далее подбираются операторы обработки данных в соответствии с их типом.

В полученных решениях порядок следования операторов изменить нельзя, иначе будет нарушен результат.

Решенная задача может встретиться как часть другой, большей задачи, поэтому ее решение следует запомнить.

Упражнения:

1. Как изменится результат работы программы obmen, если в ней поменять местами операторы, отмеченные комментариями 2 и 3? 1 и 3? 1 и 2?

2. Какой результат получится, если на вход алгоритма из третьего решения (решения с операцией xor) подать данные типа string?

Рассмотрим робот Кибик, умеющий исполнять следующий набор команд и проверок, заданных в виде функций:

сум(a,b) - возвращает сумму аргументов a и b;

раз(a,b) - возвращает разность аргументов a и b;

кв(a) - возвращает квадрат аргумента a;

пол(a) - возвращает половину аргумента a;

?() - вводит число, заданное пользователем;

!() - выводит результат;

(Н) – ª Ł ø $\begin{cases} +1, & æ^{o} Ł\ x > 0 \\ 0, & æ^{o} Ł\ x = 0, \\ -1, & æ^{o} Ł\ x < 0. \end{cases}$

Все используемые роботом числа - вещественные. Программа для робота строится в виде суперпозиции функций, т.е. вложения их друг в друга.

Пример 8.7. Пусть необходимо вычислить сумму двух заданных пользователем чисел. Тогда программа для Кибика запишется так: !(сум(?(),?())). Внимательно проверьте соответствие скобок, здесь это существенно. Если полученную программу назвать сум2=!(сум(?(),?())) и записать в библиотеку программ робота, то ее можно будет использовать в других программах наравне с командами робота. Однако в отличие от команд, выполнение сум2 вызовет исполнение программы из библиотеки, состоящей из нескольких команд.

Включение новых программ в библиотеку робота можно сравнить с обучением, т.к. с каждой новой включенной программой расширяется круг решаемых роботом задач, т.е. увеличиваются его знания и умения.

Пример 8.8. Составим для Кибика программу для вычисления выражения: 352+498-703. Здесь исходными данными являются три числа. Они могут быть любыми, поэтому в общем случае искомая программа должна находить значение выражения a+b-c. (Заметьте, что меняющиеся от одного вычисления к другому значения констант мы заменили переменными.)

Расставим скобки в программируемом выражении так, чтобы отразить порядок его выполнения: ((a+b)-c). Заменим внутренние скобки командой робота, получим: (сум(a,b)-c). Заменим внешние скобки соответствующей командой робота: раз(сум(a,b),c). Значения переменных должны быть введены, поэтому заменим их соответствующей командой. Получим: раз(сум(?(),?()),?()). Наконец, для вывода результата добавим соответствующую команду и законченная программа будет иметь вид: !(раз(сум(?(),?()),?())).

Другой вариант программы можно получить следующим образом. Расставляем скобки: (a+(b-c)). Операции в полученном выражении последовательно заменяем командами робота:

(a+раз(b,c))

сум(a,раз(b,c))

сум(?(),раз(?(),?()))

!(сум(?(),раз(?(),?()))).

Заметим, что программа для нахождения значения выражения a+b+c+d=(((a+b)+c)+d) будет иметь вид сум(сум(сум(a,b),c),d). Программы, в которых при вычислениях команды обращаются сами к себе прямо или косвенно, называются рекурсивными. О них речь пойдет далее.

В дальнейших примерах будем считать, что все необходимые данные введены и расположены в памяти робота в переменных, именами которых являются буквы латинского алфавита.

Пример 8.9. Составим для Кибика программу для вычисления значения выражения $\frac{a+b}{2}-c$. Здесь исходными данными являются числа a, b и c. В результате получаем число. Расставим в заданном выражении скобки, отражающие порядок его вычисления (((a+b)/2)-c). Заменим каждую пару скобок, начиная с внутренних, соответствующими командами робота. Последовательно получаем программу для робота. Заменяем сумму - ((сум(a,b)/2)-c), заменяем деление на 2 - (пол(сум(a,b))-c), заменяем разность - раз(пол(сум(a,b)),c).

Программу, решающую эту же задачу, можно было получить из формулы: a/2+b/2-c. Тогда последовательно получаем (((a/2)+(b/2))-c). Заменяем a/2 - ((пол(a)+(b/2))-c), заменяем b/2 - ((пол(a)+пол(b))-c), заменяем сумму - (сум(пол(a),пол(b))-c), заменяем разность - раз(сум(пол(a),пол(b)),c).

8.5. Спецификации

Из рассмотренных примеров видно, что для решения поставленной задачи можно придумать несколько программ. Вид одних определяется структурой (строением, устройством) исходных данных (задача 1), вид других - видом формулы (порядком вычисления выражения) (задача 2). В общем случае для любой задачи справедливо: либо она не имеет алгоритма ее решения для данного исполнителя, либо алгоритм существует (в последнем случае их, как правило,

оказывается несколько). Как в таком случае определить: какая программа является правильной? Существует несколько подходов к ответу на этот вопрос.

Первый подход называют тестированием. Тестирование состоит в выполнении программы со специально выбранными значениями исходных данных - тестами, для которых заранее известен результат. Если ответ, полученный программой, не совпадает с ожидаемым, то программа содержит ошибку, которую необходимо найти и исправить. Однако если тест завершился с ожидаемым результатом, мы не можем гарантировать, что также завершится любой другой тест. Более того, нельзя сказать утвердительно и о том, что этот же тест завершится успешно в случае внесения изменений в программу.

Здесь имеется аналогия со следующей ситуацией. Предположим, что мы приехали в какой-нибудь город, например в Норильск, и обнаружили, что там во всех магазинах продаются яблоки. Можно ли сказать, что так бывает всегда? Ясно, что ответ на этот вопрос отрицательный, потому что мы не можем сделать обоснованного вывода на основе результатов одной конкретной поездки.

Поэтому при тестировании используют много разных тестов. Одни тесты учитывают структуру программы, другие - вырожденные, частные случаи исходных данных. После обнаружения и исправления ошибки необходимо повторить выполнение всех тестов. В любом случае с помощью тестирования можно убедительно показать наличие ошибок в программе, но не доказать их отсутствие.

Второй подход называется верификацией. В общем случае верификация есть установление корректности программы, т.е. отсутствия в ней ошибок. Если тестирование исследует отдельные выполнения программы, то верификация анализирует свойства всех допустимых выполнений программы с помощью формальных доказательств присутствия требуемых свойств. Использование методов верификации требует хорошей математической культуры, умения пользоваться формальным аппаратом математической логики и теории доказательств.

Основная идея верификации программы состоит в том, чтобы формально доказать соответствие между текстом программы (на языке программирования) и спецификации задачи (на языке спецификации). Программа и спецификация описывают одну и ту же задачу на разных языках. Язык спецификации ориентирован на человека, язык программирования - на компьютер. Та программа, которая соответствует спецификации задачи, признается правильной.

Спецификация - четкое и формальное описание исходных данных, результатов и того, что должна делать программа для преобразования исходных данных в результат. Здесь указывается множество возможных значений исходных данных и форма их представления в программе, множество возможных значений результатов и форма их представления, а также для каких конкретных значений исходных данных какой результат получается. Последние сведения могут использоваться при тестировании и отладке программы. Кроме того, в спецификации описывается и иллюстрируется соответствующими примерами идея преобразования исходных данных в результат.

Основу спецификации составляет идея преобразования исходных данных в результат. Ее выявление является самым сложным, творческим компонентом в

программировании. Самые общие рекомендации здесь сводятся к следующему. Программирование напоминает процесс обучения. В данном случае человек обучает робота. Для того чтобы кого-то чему-либо научить, необходимо самому уметь решать подобные задачи или представлять, как они решаются. Для уяснения задачи необходимо решить задачу "вручную" (без использования компьютера) для каких-либо исходных данных. Однако решение здесь не является самоцелью, в процессе решения необходимо наблюдать за тем, какие действия ведут к цели и фиксировать их. Возможно этот процесс потребуется повторить неоднократно. Полученные частные решения для конкретных значений исходных данных необходимо обобщить на случай любых данных из множества их возможных значений. Хорошей проверкой окончания процесса уяснения задачи является устный рассказ кому-либо о найденном методе решения или словесное изложение этого метода. Только после этого можно приступать к записи алгоритма для компьютера.

Другой способ выяснения идеи решения состоит в поиске по литературе готового алгоритма. Чтобы разобраться с готовым алгоритмом, перед программированием его необходимо выполнить «вручную» (провести трассировку) этот алгоритм для различных значений исходных данных.

Часто для решения поставленной задачи нужно модифицировать существующий алгоритм или построить ему подобный, но для других типов данных.

Каждая вновь написанная программа может быть включена в библиотеку программ компьютера или робота. Этим достигается расширение системы команд робота или компьютера и тем самым осуществляется его обучение. В отличие от команд и проверок, составляющих систему команд исполнителя (СКИ), выполнение программы из библиотеки вызывает исполнение последовательности команд и проверок из СКИ или других программ.

8.6. Продолжение обучения робота Кибик

Пример 8.10. Необходимо научить робота умножать, т.е. написать программу, которая под именем умн(a,b) будет включена в библиотеку программ робота, причем умн(a,b)=a·b.

Решение. Для построения алгоритма внимательно посмотрим на СКИ робота. В ней нет команды умножения, но есть команды нахождения суммы, разности, возведения в квадрат. Попробуем найти квадрат суммы или разности. Используя формулы сокращенного умножения, получаем $(a+b)^2=a^2+2\cdot a\cdot b+b^2$. Отсюда

$$a\cdot b=((a+b)^2-a^2-b^2)/2. \quad (1)$$

Последнюю формулу можно переписать в виде:

$$a\cdot b=((a+b)^2-(a^2+b^2))/2. \quad (2)$$

Используя другую формулу сокращенного умножения, получаем

$$a\cdot b=(a^2+b^2-(a-b)^2)/2. \quad (3)$$

Построим программу для робота по формуле (1). Расставим сначала скобки, отражающие порядок выполнения операций, а затем заменим операции в скобках соответствующими командами робота. Начнем замену с внутренних скобок (заменяемые части подчеркнуты). Последовательно получаем:

$a\cdot b=((\underline{(a+b)}^2-a^2)-b^2)/2)$,

умн(a,b)=((((сум(a,b))^2-a^2)-b^2)/2),
умн(a,b)=(((кв(сум(a,b))-a^2)-b^2)/2),
умн(a,b)=(((кв(сум(a,b))-кв(a))-b^2)/2),
умн(a,b)=((раз(кв(сум(a,b)),кв(a))-b^2)/2),
умн(a,b)=((раз(кв(сум(a,b)),кв(a))-кв(b))/2),
умн(a,b)=(раз(раз(кв(сум(a,b)),кв(a)),кв(b))/2),
умн(a,b)=пол(раз(раз(кв(сум(a,b)),кв(a)),кв(b))/2).

Программы по формулам (2) и (3) строятся аналогично.

(2) умн(a,b)=пол(раз(кв(сум(a,b)),сум(кв(a),кв(b)))).

(3) умн(a,b)=пол(раз(сум(кв(a),кв(b)),кв(раз(a,b)))).

Для каждой программы спецификацией является та формула, по которой строилась программа. Если программа не соответствует той формуле, которая представлена в качестве спецификации, то в ней скорее всего содержатся ошибки, которые нужно найти и исправить. В данном случае это означает, что нужно привести в соответствие спецификацию и программу. Можно, конечно, правильной программе (2) поставить в соответствие правильную формулу (1), но это вызовет большие недоумения в процессе эксплуатации программы и чревато ошибками, возникающими из-за непонимания.

Пример 8.11. Написать программу для робота, вычисляющую y=|x|.

$$y = \begin{cases} x, & \text{если } x > 0, \\ 0, & \text{если } x = 0, \\ -x, & \text{если } x < 0. \end{cases}$$

Решение.

Сравним запись определения модуля с командой знак(a), имеющейся в системе команд робота:

$$\text{знак}(x) = \begin{cases} +1, & \text{если } x > 0 \\ 0, & \text{если } x = 0, \\ -1, & \text{если } x < 0. \end{cases}$$

Сравнив записи, можно заметить, что первая может быть получена умножением команды знак(x) на x. Так можно получить новую команду: мод(x)=умн(знак(x),x). Для решения задачи мы воспользовались не только командами робота, но и программой, полученной в предыдущем примере.

Пример 8.12. Написать программу для робота, указывающую максимальное из двух заданных чисел: max(a,b).

Решение. Для нахождения максимального значения необходимо заданные числа сравнить между собой. Однако в системе команд робота нет операций сравнения, кроме операции знак(a). Узнать, на сколько одно число отличается от другого, можно с помощью операции разность. Чтобы такое сравнение было наглядным, представим числа a и b в виде отрезков, длины которых равны a и b, и нарисуем отрезки друг по другом (рис.8.1). Примем для определенности a>b. Из рисунка видно, что длина большего отрезка больше длины меньшего отрезка на |a-b|. Если найти сумму длин отрезков a, b, |a-b|, то полученное число будет равно удвоенному максимальному.

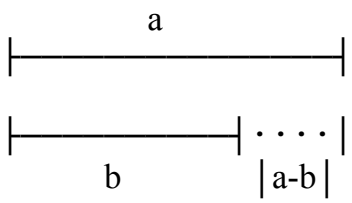


Рис. 8.1

Отсюда a+b+|a-b|=2·max. Тогда программа для робота будет иметь вид: max(a,b) = пол(сум(сум(a,b),мод(раз(a,b)))).

Пример 8.13. Написать программу для робота, определяющую минимальное из двух заданных чисел: min(a,b).

Решение. Рассуждая аналогично предыдущему, можно получить формулу, а затем в соответствии с ней написать программу:

min(a,b) = пол(раз(сум(a,b),мод(раз(a,b)))).

Упражнение. Восстановите формулу, по которой была написана эта программа.

Попытаемся найти другое решение поставленной задачи, применяя часто используемый прием построения алгоритмов, когда решение новой задачи сводится к уже решенной. К настоящему моменту мы умеем находить максимум. Исходных чисел у нас два, поэтому оставшееся будет минимальным. Отсюда следующая формула: min(a,b)=a+b-max(a,b), и программа по этой формуле: min(a,b)=раз(сум(a,b),max(a,b)).

Еще один вариант программы можно получить, если заметить, что min(a,b)=-max(-a,-b). Перепишем формулу в удобном для составления программы виде: min(a,b)=0-max(0-a,0-b).

Упражнение. Проверьте справедливость последней программы, подставив в нее числа, например: a=-5, b=9 или a=11, b=2.

Программа, построенная по последней формуле, имеет вид min(a,b)=раз(0,max(раз(0,a),раз(0,b))).

Пример 8.14. Написать программу для робота, упорядочивающую три заданных числа.

Решение. Составим спецификацию задачи.

Исходные данные: три произвольных числа. Обозначим их: a, b, c.

Результат: эти же числа, расположенные в порядке: минимальное, среднее, максимальное.

Решая задачу, будем рассуждать так. Предположим, что у нас есть готовые программы для нахождения нужных значений. Назовем их: min(a,b,c) - минимальное из трех, ср(a,b,c) - среднее из трех, max(a,b,c) - максимальное из трех. Тогда поставленную задачу можно решить с помощью, например, такой программы: !(min(a,b,c),ср(a,b,c),max(a,b,c)).

Получив это решение, мы свели сложную исходную задачу к трем более простым. Решим их последовательно.

Для нахождения минимального из трех можно использовать программу min(a,b,c)=min(min(a,b),c)). Здесь дважды использовалась написанная ранее программа нахождения минимального из двух.

Аналогично можно построить программу для нахождения максимума из трех: max(a,b,c)=max(max(a,b),c)). Но можно построить эту программу, используя ранее построенную: max(a,b,c) = раз(0, min(раз(0,a), раз(0,b), раз(0,c))).

Упражнение. Проверьте правильность работы последней программы, подставив в нее числа.

Для нахождения среднего по величине числа можно поступить, например, так: среднее=a+b+c-max(a,b,c)-min(a,b,c). В этом случае программа для робота будет иметь вид: ср(a,b,c) = раз(раз (сум (сум(a,b), c), max(a,b,c)), min(a,b,c)).

Можно также заметить, что средним будет число, являющееся максимальным из минимумов всевозможных пар, которые можно образовать из трех чисел. Всего возможны три пары: ab, ac, bc. Отсюда получаем программу ср(a,b,c)=max(min(a,b),min(a,c),min(b,c)).

Можно найти среднее как минимум из максимумов пар, тогда программа будет иметь вид ср(a,b,c)=min(max(a,b),max(a,c),max(b,c)).

В этом примере для построения программы использовался метод пошаговой детализации (метод последовательных уточнений, структурное программирование). Метод состоит в том, что исходная задача разбивается на ряд подзадач, менее сложных, но в совокупности позволяющих решить исходную сложную задачу. На основе этого разбиения составляется программа, в которой для решения подзадач используются обращения к еще ненаписанным программам (вспомогательным алгоритмам). Их можно заменить заглушками - подпрограммами, не решающими подзадачу, а возвращающими заранее известный ответ. Заглушки моделируют работу подзадачи. Используя заглушки, можно начать отладку основной программы до написания программ, реализующих подзадачи. При разработке программ для подзадач в свою очередь используется тот же прием: подзадачи разбиваются на мелкие, легко программируемые части, решающие в совокупности подзадачу. Такое разбиение продолжается до тех пор, пока в качестве подзадач не будут использованы стандартные функции и операторы языка программирования.

Этот метод облегчает построение программы, так как позволяет решать задачу по частям и отлаживать программу с помощью заглушек до завершения полного ее построения. Разбивая задачу, на каждом шаге мы отвлекаемся от рассмотрения всех деталей задачи, а учитываем только то, что необходимо для данного разбиения.

Пример 8.15. Написать программу для робота, вычисляющую $y=5(3x^2+4)^4-3(3x^2+4)^3+7(3x^2+4)^2-1/4\cdot(3x^2+4)$.

Решение. Составим спецификацию задачи.

Исходные данные: переменная x.

Результат: значение заданного выражения в переменной y.

Отвлекаясь от деталей, представим заданное выражение формулой y=(((f1(x)-f2(x))+f3(x))-f4(x)). Здесь f1(x) - вспомогательный алгоритм для вычисления выражения $5(3x^2+4)^4$, f2(x) - вспомогательный алгоритм для вычисления выражения $3(3x^2+4)^3$, f3(x) - вспомогательный алгоритм для вычисления выражения $7(3x^2+4)^2$, f4(x) - вспомогательный алгоритм для вычисления выражения $1/4\cdot(3x^2+4)$.

Программа для робота будет иметь вид выр(x) = раз(сум(раз(f1(x), f2(x)), f3(x)), f4(x)).

Написав, например, заглушки: f1(x)=сум(1,1), f2(x)=сум(1,1), f3(x)=сум(1,1), f4(x)=сум(1,1), можно проверить работоспособность написанной программы.

Продолжим детализацию, напишем программу для $f1(x)=5(3x^2+4)^4$. Заметим, что выражение в скобках повторяется во всех вспомогательных алгоритмах, поэтому разработаем для него отдельную программу.

$ff(x) = 3x^2+4 = ((3\cdot(x^2))+4) =$ сум(умн(3,кв(x)),4).

Четвертую степень можно найти как кв(кв(x)).

Тогда f1(x)=умн(5,кв(кв(ff(x)))). Сейчас заглушку f1(x) можно заменить написанной программой и проверить правильность работы всей программы.

Третью степень можно найти как умн(кв(x),x).

Тогда f2(x)=умн(3,умн(кв(ff(x)),ff(x))), а f3(x)=умн(7,кв(ff(x))). Одну четвертую можно представить как пол(пол(x)), тогда f4(x)=пол(пол(ff(x))).

Все вспомогательные алгоритмы разработаны. Сейчас ими можно заменить заглушки и проверить работу всей программы. Можно поступить иначе, подставив написанные вспомогательные алгоритмы в основной. Будет получена длинная запись с большим количеством скобок, работать с которой нужно очень осторожно, чтобы не удалить нужную и не подставить лишнюю скобку.

Упражнение. Подставьте все вспомогательные алгоритмы в основной алгоритм решения задачи.

8.7. Линейные алгоритмы на Паскале

Запишем пример 8.15 в виде фрагмента программы на Паскале. Повторяющуюся часть выражения нужно вычислить отдельно, чтобы не повторять одинаковые вычисления. Затем можно вычислить все выражение. В этом примере операторы Паскаля запишутся последовательно друг за другом и выполнять их нужно в порядке записи. Переставить местами эти операторы без нарушения результата не удается.

Для хранения повторяющейся части выражения можно также использовать переменную y, потому что при выполнении оператора присваивания сначала вычисляется значение выражения, стоящее справа, а затем вычисленное значение пересылается в переменную, указанную слева.

```
write('Введите x ');
readln(x);
y:=3*sqr(x)+4;
y:=5*sqr(sqr(y))-3*sqr(y)*y+7*sqr(y)-0.25*y;
write('Ответ ',y)
```

При выполнении второго оператора присваивания потребовалось выполнить 10 операций.

Для сокращения количества операций применим схему Горнера. Вынесем у первых двух слагаемых общий множитель за скобки: $\underline{5y^4-3y^3}+7y^2-1/4\cdot y = (5y-3)y^3+7y^2-0{,}25y$.

Еще раз у первых двух слагаемых вынесем общий множитель за скобки: $\underline{(5y-3)y^3+7y^2}-0{,}25y=((5y-3)y+7)y^2-0{,}25y$.

Еще раз проделаем ту же процедуру и окончательно получим $((5y-3)y+7)y^2-0{,}25y = ((5y-3)y+7)y-0{,}25)y$.

Для вычисления полученного выражения нужно всего лишь семь операций. Соответствующий фрагмент программы показан ниже:

```
write('Введите x ');
```

```
readln(x);
y:=3*sqr(x)+4;
y:=((5*y-3)*y+7)*y-0.25)*y;
write('Ответ ',y)
```

Запомним, что для решения одной и той же задачи можно построить разные алгоритмы. Они отличаются друг от друга эффективностью, т.е. количеством используемых операций и переменных. В рассмотренном примере с помощью схемы Горнера удалось построить алгоритм с меньшим количеством операций, не вводя дополнительных переменных. Однако это возможно не всегда. Чаще всего выигрыш во времени выполнения алгоритма влечет дополнительный расход памяти (вводятся дополнительные переменные) и наоборот.

Пример 8.16. Присвоить целой переменной h третью от конца цифру в записи положительного целого числа k (например, если k=130985, то h=9).

*Решение***.** Здесь задано произвольное целое число, которое в компьютере представляется в виде последовательности нулей и единиц. Над ним разрешено выполнение арифметических операций и стандартных функций. Это представление отличается от представления чисел, привычного для человека. В последнем случае число - это последовательность записанных друг за другом цифр. Его можно рассматривать как число, тогда над ним выполняются арифметические операции и стандартные функции. Можно это же число рассматривать и как последовательность символов, тогда к нему применимы операции «вырезки» отдельных символов, замены их другими символами, вставки новых символов и другие операции, разрешенные над строками символов. Такие операции над машинным представлением числа невозможны. Однако приведенные рассуждения приводят к мысли изменить форму представления числа, использовав для этого процедуру str(число,строка), а затем функцию copy(строка,позиция,длина). Получим:

```
var     k:integer;      { заданное число }
        h:string;       { третья цифра заданного числа }
        s:string;       { число, представленное в виде строки символов }
begin   str(k,s);       { преобразовали число в строку 13098 -> '13098' }
        h:=copy(s,length(s)-2,1) { выделили нужную цифру }
end.
```

Возможно другое решение. Представим исходное число в многочленной форме: $1\cdot10^5+3\cdot10^4+0\cdot10^3+9\cdot10^2+8\cdot10+5$. Такое представление числа подсказывает, что последнюю цифру числа можно найти как остаток от деления на 10, т.е. k mod 10. Это упрощает исходную задачу, сводя ее к задаче изменения числа таким образом, чтобы третья цифра числа стала последней, выделять которую мы научились. Последняя задача сводится к переносу запятой в числе на два разряда влево, что равносильно уменьшению числа в 100 раз, полученная в результате такого переноса дробная часть не нужна и ее можно отбросить. Описанную работу можно выполнить, воспользовавшись операцией: k div 100, а исходную задачу можно решить так: h:=k div 100 mod 10.

Заметьте, что в данном решении использован прием сведения задачи к решенной ранее.

Обобщим полученное решение на случай выделения i-й цифры числа:

1) переносим запятую влево на i-1 знак (i-я цифра становится последней в числе);

2) находим последнюю цифру как остаток от деления на 10. Схема решения такова: k div 10^{i-1} mod 10.

Проверим работу схемы при решении задачи нахождения четвертой цифры в числе k=13. Получим: k div 10^3 mod 10 =0. Действительно, если записать число с незначащими цифрами, то получим 0013, откуда следует ответ на поставленную задачу.

Упражнение. Обобщите первое решение (с преобразованием формы представления числа) на случай выделения произвольной цифры.

Рассмотрим задачу нахождения первой цифры числа. Для применения рассмотренной выше схемы нужно знать количество цифр в числе. Их можно сосчитать (что мы научимся делать в дальнейшем), а можно воспользоваться тем фактом, что количество цифр в положительном числе, большем единицы, равно целой части десятичного логарифма этого числа плюс единица. Здесь целая часть - наибольшее целое, не превосходящее данное. Таким образом, для нахождения первой цифры числа можно предложить такую схему: число div $10^{]\ \lg\ \text{число}\ [}$. Здесь] lg число[обозначает целую часть логарифма числа.

Упражнение. Найдите первую цифру числа в решении, основанном на преобразовании его формы представления.

Пример 8.17. Присвоить целой переменной d первую цифру из дробной части положительного вещественного числа x. Например, если x=32,597, то d=5.

Решение. Заметим, что решение этой задачи можно получить, если умножить заданное число на 10. В этом случае задача сведется к выделению последней цифры полученного произведения. Нужно еще учесть, что операция mod применима к целому числу, поэтому произведение нужно преобразовать к целому виду, например, отбросив дробную часть. Получаем оператор, решающий задачу: d:=trunc(x*10) mod 10.

Возможно решение с помощью преобразования числа в строку символов. В этом случае в строке нужно найти позицию точки и выделить следующую за ней цифру. Получаем такой фрагмент:

```
var     x:real; { заданное число }
        d:integer; { первая цифра дробной части заданного числа }
        s:string; { число, представленное в виде строки символов }
begin   str(x,s); { преобразовали число в строку 32.597 -> '32.597' }
        d:=copy(s,pos('.',s)+1,1) { выделили нужную цифру }
end.
```

Чтобы выделить i-ю цифру дробной части можно предложить следующую схему: trunc(число * 10^i) mod 10. Здесь нужно помнить, что правильный ответ будет получен только для хранимых в памяти компьютера цифр дробной части.

Упражнение. Напишите фрагмент программы для выделения i-й цифры дробной части числа с помощью преобразования его в строку символов.

Пример 8.18. Целой переменной s присвоить сумму цифр трехзначного целого числа k.

Решение. Воспользуемся схемой для общего случая, тогда третья цифра может быть найдена так: k div 100 mod 10, вторая - k div 10 mod 10, последняя - k div 1 mod 10. Обратите внимание, что получились одинаково устроенные выражения, отличающиеся только первым делителем. Это значит, что для нахождения каждой цифры нужно выполнить одинаковые операции, что является основой для построения циклов (рассматриваются дальше). Впрочем, последняя цифра может быть определена как k mod 10, а первая - как k div 100. Тогда оператор присваивания, решающий задачу, будет иметь вид s:=k div 100+k div 10 mod 10+k mod 10.

Упражнение. Решите эту задачу с помощью преобразования числа в строку.

Пример 8.19. Идет k-я секунда суток. Написать операторы, выясняющие, сколько полных часов (h) и полных минут (m) прошло к данному моменту.

Решение. Вспомним, что 1 час равен 60 минутам,
1 минута равна 60 секундам.

Пусть, например, k=13257. Узнаем сколько полных часов прошло: h = 13257 div (60*60) = 3. После этой операции осталось: 13257 mod (60*60)=2457 секунд. Для выделения полных минут остаток разделим на 60: m=2457 div 60=40. Остаток от деления 2457 на 60 покажет количество оставшихся секунд: 2457 mod 60=57. Полностью решение можно записать так:

```
var   k:integer; { исходное данное - количество секунд }
      h:integer; { прошло количество часов }
      m:integer; { прошло количество минут }
      s:integer; { прошло количество секунд }
begin h:=k div 3600;
      m:=k mod 3600 div 60;
      s:=k mod 3600 mod 60
end.
```

Обобщим решение на следующий случай (предложен Г.Остером):

1 кукаляка равна A мукака,
1 мукака равна B бисяка,
1 бисяка равна C пампукским хрюрям,
1 пампукская хрюря равна D мамурикам.

Имеется E мамуриков. Требуется определить, сколько полных кукаляк(v), мукак(w), бисяк(x) и пампукских хрюрь(y) имеется.

Поскольку структура (устройство) данных решаемой задачи похожа на структуру данных предыдущей задачи, то и их решения (алгоритмы) должны совпадать, так как это фактически одна и та же задача.

На основании этого последовательно получаем:

v=E div (A*B*C*D),
w=E mod (A*B*C*D) div (B*C*D),
x=E mod (A*B*C*D) mod (B*C*D) div (C*D),
y=E mod (A*B*C*D) mod (B*C*D) mod (C*D) div D,
z=E mod (A*B*C*D) mod (B*C*D) mod (C*D) mod D.

Можно избавиться от повторяющихся частей формул, тогда получим:

v=E div A*B*C*D, z=E mod A*B*C*D;
w=z div B*C*D, z=z mod B*C*D;

x=z div C*D, z=z mod C*D;

y=z div D, z=z mod D.

Здесь выявляются повторяющиеся операторы, которые можно использовать в цикле, обрабатывающем таблицу исходных данных для произвольного количества строк.

***Упражнения*:**

1. Старинная английская система денежных единиц состояла из фунтов, шиллингов (12 шиллингов = 1 фунту) и пенсов (20 пенсов = 1 шиллингу). Напишите фрагмент программы, выясняющий сколько полных фунтов, шиллингов и пенсов можно получить, если имеется A пенсов.

2. Старинными русскими денежными единицами являются: 1 рубль - 100 копеек, 1 гривна - 10 копеек, 1 алтын - 3 копейки, 1 полушка - 0,25 копейки. Имеется A копеек. Написать фрагмент программы для представления имеющейся суммы в рублях, гривнах, алтынах и полушках. Как свести решение этой задачи к решению предыдущей?

Вернемся к решению задачи 8.19. Заданное количество секунд можно представить так: 13257=3·60·60+40·60+57. Такое представление эквивалентно представлению числа в шестидесятеричной системе счисления. Веса разрядов можно изменить, тогда получим новую систему счисления.

Упражнение. В системе счисления «8-6-4-2» для записи чисел в младшем разряде используются цифры 0;1, в следующем разряде - цифры: 0;1;2;3, в следующем - 0;1;2;3;4;5, в четвертом разряде - 0;1;2;3;4;5;6;7. Написать фрагмент программы перевода числа A из десятичной системы счисления в систему «8-6-4-2».

9. ВЕТВЯЩИЕСЯ АЛГОРИТМЫ

Ветвящимся называется алгоритм, в котором в зависимости от выполнения некоторого условия выполняется та или иная группа действий. Условие задается логическим выражением и принимает два значения: false (ложь) или true (истина). На Паскале ветвящиеся алгоритмы записываются с помощью условных операторов. Полный условный оператор записывается так:

```
if условие
then оператор1
else оператор2.
```

Здесь оператор 1 и/или оператор 2 может быть любым, но только одним оператором Паскаля. Точка с запятой перед else не ставится, потому что в противном случае она окажется внутри условного оператора. Оператор 1 будет выполнен, когда условие будет истинным, в противном случае будет выполнен оператор 2. Если после then или else необходимо выполнить несколько операторов, то последние объединяются в один составной оператор операторными скобками begin end. Общая структура составного оператора такова: *begin оператор1;оператор2;...;операторN end*.

Здесь, как обычно, точка с запятой используется для разделения операторов. Все операторы выполняются в порядке их записи.

В Паскале используется также сокращенный условный оператор, в котором фраза else и оператор 2 опущены:

```
if условие
then оператор1.
```

Здесь оператор 1 выполняется, если условие истинно. Если условие ложно, то выполняется оператор, следующий за сокращенным условным. Введение сокращенного условного оператора в ряде случаев приводит к неоднозначностям в записи программы.

Рассмотрим, например, случай, когда оператор 1 в полном условном операторе является сокращенным условным. Получим конструкцию:

```
if условие1
then    if условие2
        then оператор1
        else оператор2.
```

В записи этой конструкции непонятно, к какому оператору - первому или второму, относится фраза else. Поэтому в Паскале принято правило, устраняющее такие неоднозначности. В соответствии с ним фраза else относится к ближайшей фразе then. Следовательно, правильная запись обсуждаемой выше конструкции будет такой:

```
if условие1
then    begin  if условие2
                  then оператор1
        end
else оператор2,
```

или такой:

```
if условие1
then    if условие2
        then оператор1
        else
else оператор2.
```

Здесь после первой фразы else находится пустой оператор. Последние записи являются также примерами вложенных условных операторов.

Рассмотрим запись условного оператора, в котором действие выполняется только при ложном условии. Правильные записи здесь будут такими:

```
if условие                    if not условие
   then          или              then оператор.
   else оператор
```

Назовем ветвью последовательность условий и операторов, которые будут пройдены при выполнении алгоритма для конкретных значений исходных данных. Заметьте, что во всех рассмотренных выше условных операторах, которые можно рассматривать как запись простейших ветвящихся алгоритмов, количество ветвей равно количеству условий плюс единица. Рассмотрим еще пример:

```
if условие1
then    if условие2
        then оператор1
        else оператор2
else    if условие3
        then оператор3
        else оператор4
```

В этом примере 3 (три) условия, поэтому, в соответствии с предложенной формулой, количество ветвей будет равно 4. Запишем все ветви этого алгоритма в таблице.

Ветви алгоритма

Ветвь	**Условие1**	**Условие2**	**Условие3**	**Оператор**
1	true	true		1
2	true	false		2
3	false		true	3
4	false		false	4

Докажем, что для произвольного ветвящегося алгоритма, построенного с помощью вложения других ветвящихся алгоритмов, количество ветвей равно количеству условий плюс единица.

Воспользуемся методом математической индукции. Для количества условий, равного единице, количество ветвей равно двум. Этот факт следует из рассмотренных выше ветвящихся алгоритмов (условных операторов). Предположим, что в ветвящемся алгоритме количество условий равно n, а количество ветвей - n+1. Докажем, что при добавлении еще одного условия количество ветвей станет n+2. В самом деле, добавление условия вызывает замену в ветвящемся алгоритме одной ветви двумя новыми, поэтому количество ветвей станет: n+1-1+2=n+2 или (n+1)+1, где n+1 - количество условий. Таким образом, утверждение доказано.

Ветвящийся алгоритм может быть построен последовательным соединением других ветвящихся алгоритмов. Покажем, что и в этом случае формула для подсчета количества ветвей остается той же. Рассмотрим
пример:

```
if условие1          Первая часть
then  оператор1      ветвящегося
else оператор2;      алгоритма
if условие2          Вторая часть
then  оператор3      ветвящегося
else оператор4       алгоритма
```

В соответствии с доказанным ранее, каждая часть приведенного ветвящегося алгоритма имеет по две ветви, так как содержит по одному условию. Воспользуемся основным принципом комбинаторики, согласно которому, если объект А можно выбрать m способами, а после каждого такого выбора объект В можно выбрать n способами, то выбор пары (А,В) в указанном порядке можно осуществить m·n способами. В нашем случае первая часть ветвящегося алгоритма содержит две ветви, после выбора любой из этих ветвей мы вынуждены проходить любую из двух ветвей второй части. Следовательно, всего ветвей в этом алгоритме: 2·2=4. К этому же факту можно прийти путем преобразования исходного алгоритма. Для этого внесем вторую часть алгоритма в каждую ветвь первой части. Преобразованный алгоритм будет иметь вид:

```
if условие1
then      begin  оператор1;
```

```
                if условие2
                then оператор3
                else оператор4
        end
else    begin  оператор2;
                if условие2
                then оператор3
                else оператор4
        end.
```

В преобразованном алгоритме три условия, следовательно, по ранее доказанному, - четыре ветви.

В общем случае в ветвящемся алгоритме могут произвольным образом сочетаться ветви, полученные вложением одного ветвящегося алгоритма в другой, и ветви, полученные последовательным соединением двух и более других ветвящихся алгоритмов. В этом случае для определения общего количества ветвей исходный алгоритм разбивается на части, относящиеся к одному из рассмотренных видов ветвящихся алгоритмов. Для каждой части применяют установленные ранее правила подсчета, а затем используют основные правила комбинаторики. Одно из них было рассмотрено ранее. Его применение относится к случаю последовательного соединения ветвящихся алгоритмов.

Второе правило: если некоторый объект А можно выбрать m способами, а другой объект В - n способами, то выбор «либо А, либо В» можно осуществить m+n способами. Это правило применяют тогда, когда надо узнать количество ветвей в двух сложно построенных ветвях ветвящегося алгоритма. Рассмотрим пример.

```
if условие1
then    begin
```

```
if условие2
then    if условие3
     then оператор1
     else        if условие4
                 then оператор2
                 else оператор3
else оператор4;
```

часть А

```
if условие5
then оператор5
else оператор6
```

часть В

```
        end
else    begin
                if условие6
                then оператор7
                else оператор8;
                if условие7
                then оператор9
                else оператор10
        end.
```

Операторы, которые нужно выполнять в случае истинности условия 1, можно разбить на две части A и B, которые выполняются последовательно друг за другом. Часть A построена как вложенные друг в друга ветвящиеся алгоритмы. В нее входят три условия, поэтому количество ветвей в части A равно 4. В части B две ветви. Общее количество ветвей найдем в соответствии с правилом комбинаторики: 4·2=8.

Операторы, которые нужно выполнить в случае ложности условия 1, представляют два последовательно записанных условных оператора. Поэтому общее количество ветвей здесь равно 2·2=4.

Общее количество ветвей в примере найдем, применив последнее из рассмотренных правил комбинаторики - правило сложения: 8+4=12.

В этом можно убедиться, преобразовав алгоритм, внеся операторы в соответствующие ветви. Часть B вставляется к операторам 1, 2, 3 и 4. Условный оператор с условием 7 - к операторам 7 и 8. Получаем следующую преобразованную схему:

```
if условие1
then    if условие2
        then    if условие3
                then    begin  оператор1;
                               if условие5
                               then оператор5
                               else оператор6
                        end
                else    if условие4
                        then    begin  оператор2;
                                       if условие5
                                       then оператор5
                                       else оператор6
                                end
                        else    begin  оператор3;
                                       if условие5
                                       then оператор5
                                       else оператор6
                                end
        else    begin  оператор4;
                       if условие5
                       then оператор5
```

```
                    else оператор6
              end
else    if условие6
        then  begin  оператор7;
                     if условие7
                     then оператор9
                     else оператор10
              end
        else  begin  оператор8;
                     if условие7
                     then оператор9
                     else оператор10
              end.
```

В преобразованной схеме 11 условий, следовательно, 12 ветвей.

Подсчет количества ветвей алгоритма не является самоцелью. При тестировании алгоритмов необходимо пройти каждую их ветвь хотя бы по одному разу, а для этого необходимо уметь подсчитывать количество ветвей в алгоритмах.

9.1. Построение условий

Синонимом условия является *логическое выражение*. Это логическая константа, логическая переменная, логическая стандартная функция, их комбинация с логическими операциями (not, and, or) и круглыми скобками или два нелогических выражения, соединенные операциями сравнения.

Простейшим и наиболее распространенным случаем условия является сравнение двух выражений одного типа. Имеется шесть операций сравнения:

<	меньше	>=	больше или равно
<=	меньше или равно	>	больше
=	равно	<>	неравно

Общий вид условия представляется так:

выражение1 знак_операции_сравнения выражение2.

Здесь выражения 1 и 2 должны быть одного типа, иначе сравнение завершится АВОСТом. Исключение допускается лишь для данных целого и вещественного типа. Результатом операции является «истина» или «ложь».

Для практического программирования важно знать, как записываются наиболее часто используемые условия, и уметь по известным образцам записывать новые.

Заметим, что в виде условий и/или вложенных ветвящихся алгоритмов записываются алгоритмы распознавания, которые позволяют распознать, подходит ли некоторый объект под заданное определение(понятие).

Рассмотрим примеры:

Пример 9.1. Проверить, является ли число a четным.

Решение. Это задание означает, что нужно написать условие истинное, если число a является четным, и ложное в противном случае. Вспомним определение четного числа. Число, делящееся на 2 нацело (без остатка), называется четным. Отсюда следует, что если при делении числа a на 2 остаток будет равен нулю, то число будет четным. Условие на Паскале можно записать одним из предложенных ниже вариантов.

a mod 2=0,
a-(a div 2)*2=0,
a-trunc(a/2)*2=0.

Пример 9.2. Проверить, является ли число a кратным числу b.

Решение. Определение: число a кратно числу b, если a делится на b нацело (без остатка). Отсюда: a mod b=0, a-(a div b)*b=0 или a-trunc(a/b)*b.

Упражнение. Сравните решения первого и второго примеров, найдите общие черты и различия, сделайте выводы.

Пример 9.3 Проверить, имеет ли действительные корни квадратное уравнение $ax^2+bx+c=0$ ($a\neq0$).

Решение. Определение: квадратное уравнение имеет действительные корни, если его дискриминант неотрицательный. Это определение можно записать на Паскале так: sqrt(b*b-4*a*c)>=0 или sqrt(sqr(b)-4*a*c)>=0.

Пример 9.4. Проверить, лежит ли точка с координатами (x,y) вне круга радиуса r с центром в начале координат (точка 0,0).

Решение. Определение: точки, лежащие на окружности радиуса r с центром в точке (x_c,y_c), удовлетворяют уравнению $(x-x_c)^2+(y-y_c)^2=r^2$. Поскольку нас интересуют точки, лежащие вне круга, следовательно, их расстояние до центра должно быть больше радиуса. С учетом того, что центр круга лежит в точке (0,0), искомое условие запишется так: sqr(x)+sqr(y)>sqr(r).

Пример 9.5. Проверить, является ли натуральное n полным квадратом.

Решение. Определение: число является полным квадратом, если квадратный корень из него является числом целым. У целого числа отсутствует дробная часть, поэтому если извлечь квадратный корень из заданного числа, отбросить у него дробную часть и возвести результат в квадрат, то полученное значение может совпасть с исходным только тогда, когда исходное значение было целым. Исходя из этого получаем условие sqr(trunc(sqrt(n)))=n.

Пример 9.6. Проверить, принадлежит ли действительное число x отрезку [-1;1].

Решение. Определение: число принадлежит отрезку, если выполняется неравенство $-1\le x\le1$ или $|x|\le1$. Соответствующее условие на Паскале запишется так: abs(x)<=1 или (-1<=x) and (x<=1).

Пример 9.7. Проверить, принадлежит ли действительное число x отрезку [0;1].

Решение. Определение: число принадлежит отрезку, если выполняется неравенство $0\le x\le1$. Для того чтобы записать его так же, как предыдущий случай, найдем середину интервала (1+0)/2=0,5. Затем из всех частей неравенства вычтем эту середину: $-0,5\le x-0,5\le0,5$ или $|x-0,5|\le0,5$. Условие на Паскале запишется так: abs(x-0.5)<=0.5 или (0<=x) and (x<=1).

Пример 9.8. Проверить, принадлежит ли действительное число x отрезку [a;b].

Решение. Для истинности этого условия необходимо, чтобы выполнялось неравенство $a\le x\le b$ (*). Найдем координату середины отрезка: $a+(b-a)/2 = (2\cdot a+b-a)/2 = (a+b)/2$. Вычтем полученное значение из всех частей неравенства (*), получим $a-(a+b)/2\le x-(a+b)/2\le b-(a+b)/2$ или $-(b-a)/2\le x-(a+b)/2\le(b-a)/2$. Отсюда $|x-(a+b)/2|\le(b-a)/2$.

Запись на Паскале: abs(x-(a+b)/2)<=(b-a)/2 или (a<=x) and (x<=b).

9.2. Высказывания и их запись на Паскале

Высказывание - это повествовательное предложение, которое либо истинно, либо ложно. Например, высказывание «Язык Паскаль разработал Блез Паскаль» является ложным, а высказывание «Язык Паскаль разработал Никлаус Вирт»- истинным.

Не всякое предложение является высказыванием. Не являются высказываниями такие предложения: «Суп - вкусное блюдо», «Программирование - интересный предмет», потому что нет и не может быть единого мнения о том, истинны эти предложения или ложны.

Высказывания на Паскале обозначаются логическими переменными. К ним возможно применение логических операций.

Пример 9.9. Записать на Паскале выражение «Возраст человека заключен между 18 и 35 годами». Найти его отрицание и записать его словами русского языка.

Решение. Пусть переменная x обозначает возраст человека. Тогда заданное высказывание может быть записано так: (18<=x) and (x<=35).

Для нахождения отрицания воспользуемся законами де Моргана. not[(18<=x) and (x<=35)]=not(18<=x) or not(x<=35)=(18>x) or (x>35). Отрицание: «Неверно, что возраст человека x заключен между 18 и 35 годами»; - «Возраст человека меньше 18 или больше 35 лет»; - (18>x) or (x>35).

Пример 9.10. Записать на Паскале высказывание «Неверно, что треугольник ABC прямоугольный и равнобедренный».

Решение. Обозначим: P - треугольник ABC прямоугольный; R - треугольник ABC равнобедренный. Тогда высказывание запишется так: not(P and R).

Применяя закон де Моргана, получаем not P or not R. Соответствующее высказывание: «Треугольник ABC не прямоугольный или не равнобедренный».

Пример 9.11. Записать на Паскале высказывание «Неверно, что число 9 четное или простое».

Решение. Обозначим Ч - «Число 9 четное», П - «Число 9 простое». Тогда исходное высказывание запишется на Паскале так: not(Ч or П). Применяя закон де Моргана, получаем not Ч and not П, что соответствует высказыванию: «Число 9 нечетное и непростое».

Пример 9.12. Записать на Паскале высказывание «Неверно, что каждое из чисел m и n четно».

Решение. Если число m четное, то выполняется условие: m mod 2=0. Точно также, если число n четное, то выполняется условие: n mod 2=0. Учитывая эти факты, заданное высказывание можно записать так: not((m mod 2=0) and (n mod 2=0)). Применяя закон де Моргана, получаем not(m mod 2=0) or not(n mod 2=0) или (m mod 2<>0) or (n mod 2<>0). Последнее условие соответствует высказыванию: «Числа m или n нечетны».

Пример 9.13. Записать на Паскале высказывание «Неверно, что хотя бы одно из чисел - r и s - простое».

Решение. Обозначим R высказывание «r - простое число». S - «s - простое число». Тогда заданное высказывание запишется на Паскале так: not(R or S)=not R and not S. Последнее условие соответствует высказыванию: «Неверно, что R и S - простые числа».

Упражнение. Докажите, что отрицанием высказывания «Я не высплюсь или опоздаю на занятия» является высказывание «Я высплюсь и не опоздаю на занятия».

Пример 9.14. Записать условие истинное, если x принадлежит отрезку [2;5] или [-1;1]. Затем найти его отрицание.

Решение. Это условие на Паскале может быть записано так: ((2<=x) and (x<=5)) or abs(x)<=1. Отрицание этого высказывания можно найти используя законы де Моргана: (x<-1) or (1<x) and (x<2) or (x>5).

Упражнение. Проверьте записанное отрицание.

Пример 9.15. Записать на Паскале высказывание «Каждый из трех друзей: Ваня, Коля или Петя, имеет карманные деньги».

Решение. Обозначим высказывание «Ваня имеет b рублей карманных денег» - b. «Коля имеет k рублей карманных денег» - k. «Петя имеет p рублей карманных денег» - p. Тогда заданное высказывание запишется так: (b>0) and (k>0) and (p>0).

Пример 9.16. Записать на Паскале высказывание «Хотя бы один из трех друзей имеет карманные деньги».

Решение. В обозначениях предыдущего примера заданное условие будет иметь вид (b>0) or (k>0) or (p>0).

Пример 9.17. Записать на Паскале высказывание «Ни один из друзей не имеет карманных денег».

Решение. В обозначениях примера 9.15 заданное условие запишется так: (b<=0) and (k<=0) and (p<=0).

Пример 9.18. Записать на Паскале высказывание «Хотя бы один из трех друзей не имеет карманных денег».

Решение. В обозначениях примера 9.15 заданное условие запишется так: (b<=0) or (k<=0) or (p<=0).

Упражнение. Условия упражнений 9.15 - 9.18 связаны друг с другом в том смысле, что они образуют пары: условие и его отрицание. Найдите эти пары.

Пример 9.19. Записать на Паскале высказывание «Только один из друзей имеет карманные деньги».

Решение. В обозначениях примера 9.15 заданное условие запишется так: ((b>0) and (k<=0) and (p<=0)) or ((b<=0) and (k>0) and (p<=0)) or ((b<=0) and (k<=0) and (p>0)).

Пример 9.20. Записать на Паскале высказывание «Целые n и k имеют одинаковую четность».

Решение. Числа имеют одинаковую четность, если их остатки от деления совпадают, поэтому n mod 2 = k mod 2.

Можно воспользоваться стандартной функцией odd(i): odd(n)=odd(k).

Пример 9.21. Записать на Паскале высказывание «Числа x,y,z равны между собой».

Решение. Заданное высказывание можно записать так: (x=y) and (x=z).

Упражнение. Приведите другие варианты записи этого высказывания. Сколько всего вариантов записи возможно?

Пример 9.22. Записать на Паскале высказывание «Из чисел x,y,z только два равны между собой».

Решение. Вариант записи: (x=y) and (x<>z) or (x=z) and (x<>y) or (y=z) and (x<>y).

Упражнение. Приведите другие варианты записи этого высказывания. Сколько всего вариантов записи возможно?

Пример 9.23. Записать на Паскале высказывание «Цифра три входит в десятичную запись трехзначного целого числа».

Решение. Вариант записи: (k div 100=3) or (k div 10 mod 10=3) or (k mod 10=3).

9.3. Построение ветвящихся алгоритмов

Пример 9.24. Найти максимальное из двух заданных значений.

Составим спецификацию. Дано: a,b - вещественные числа. Получить: m=max(a,b).

Решение. Достаточно сравнить заданные числа. Если в результате сравнения большим числом окажется a, то его присвоить переменной m. В противном случае переменной m присваиваем число b. Получаем такое решение:

```
program vet1;
var     a,b:real; { исходные числа }
        m:real; { результат: max(a,b) }
begin   write('Введите два числа ');
        readln(a,b);
        if a>b
        then m:=a     { 1 }
        else m:=b;    { 2 }
        write('Максимальное из ',a,' и ',b,'->',m)
end.
```

В этой программе две ветви. Они отмечены с помощью комментариев { 1 } и { 2 }. При тестировании нужно по каждой из этих ветвей пройти хотя бы по одному разу. Для прохождения по первой ветви можно выбрать, например, следующий набор исходных данных: a=3, b=2. В этом случае получим в качестве ответа m=3. Для прохождения по второй ветви можно выбрать, например, следующий набор данных: a=2, b=6. В этом случае получим ответ: m=6.

Задачу 9.24 можно было бы решить и так (приводится фрагмент программы):

```
if a<=b
then m:=b
else m:=a.
```

Однако это решение принципиально не отличается от предыдущего. Читателю предлагается самостоятельно оформить это решение в виде программы на Паскале и установить для нее тесты.

Другое решение задачи 1 можно получить, если поступить так. В начале переменной m присвоим произвольно одно из чисел, например a. Затем, чтобы получить ответ, сравним переменную m с переменной b. Запишем это решение на Паскале:

```
m:=a;
if m<b
then m:=b
```

В этом решении также две ветви, которые могут быть пройдены следующими тестами:

1) a=3, b=2, в этом случае условие m<b ложно;

2) a=2, b=6, в этом случае условие m<b истинно.

Последнее решение опирается на первое. В самом деле, формально можно записать так: m:=a; m:=max(m,a).

Заметим, что можно было найти максимальное из двух, не прибегая к ветвящемуся алгоритму: m:=(a+b+abs(a-b))/2.

Упражнение. Напишите фрагменты программы на Паскале, которая найдет минимальное из двух.

Пример 9.25. Написать программу, находящую максимальное из трех заданных вещественных чисел a,b,c.

Решение. Составим спецификацию. Дано: a,b,c - вещественные числа. Получить: m=max(a,b,c).

Первый вариант решения можно получить, применив к решению данной задачи решение задачи 9.24. Сначала найдем максимум из двух произвольных чисел, а затем максимум из найденного и оставшегося третьего числа.

Первый шаг решения.

```
if a>b
then { max(a,c) }
else { max(b,c) }.
```

Сейчас исходная задача упростилась и свелась к двум, решать которые мы уже умеем.

Второй шаг решения. { max(a,c) }

```
if a>c
then m:=a
else m:=c.
```

Третий шаг решения. { max(b,c) }

```
if b>c
then m:=b
else m:=c.
```

Сейчас можно собрать решение исходной задачи из отдельных частей:

```
if a>b
then      if a>c
          then m:=a     { 1 }
          else m:=c     { 2 }
  else    if b>c
          then m:=b     { 3 }
          else m:=c.    { 4 }
```

Здесь применяется метод пошаговой детализации (см. стр.49). Исходная задача разбивается на ряд частей (подзадач). Каждая подзадача представляет собой менее сложную задачу, которая так же подвергается разбиению на подзадачи. На каждом шаге подзадачи представляются с помощью одной из трех основных алгоритмических структур: линейной, ветвящейся или циклической. Разбиение продолжается до тех пор, пока не получатся элементарные задачи, каждая из которых может быть реализована одним из операторов алгоритмического языка. На

каждом шаге разбиений все полученные подзадачи могут быть представлены вызовами вспомогательных алгоритмов (процедур и функций) и протестированы до завершения всей разработки.

В этом методе любой алгоритм получается как суперпозиция (вложение) трех названных выше структур. Здесь отпадает надобность в использовании оператора «go to», который запутывает алгоритм, делает трудным его понимание, тестирование и сопровождение. Без использования этого оператора процесс построения не сильно усложняется, зато алгоритмы получаются наглядными, хорошо читаемыми, в них меньше вероятность допустить ошибки, легче их обнаружить и исправить. В дальнейшем все алгоритмы будут строиться с использованием этого метода.

Вернемся к решению 1 задачи 9.25. В этом решении четыре ветви. Для тестирования потребуется как минимум четыре теста, чтобы пройти по каждой ветви алгоритма хотя бы по одному разу.

1-я ветвь: a=3, b=2, c=1; m=3 (a>b и a>c).

2-я ветвь: a=3, b=2, c=4; m=4 (b<a<=c).

3-я ветвь: a=3, b=6, c=1; m=6 (a<=b и c<b).

4-я ветвь: a=3, b=4, c=5; m=5 (a<=b<=c).

Формально построенное решение можно записать так: m=max(max(a,b),c).

Упражнение. Постройте другие решения, основанные на этом же принципе.

Второе решение можно получить так. Предположим, что максимальным из трех заданных величин является a. Запишем ее в переменную m. Затем последовательно сравним m с b и с и, если окажется, что m меньше значения какой-либо из переменных, то это значение присвоим m. Это решение основывается на втором варианте решения предыдущей задачи.

```
m:=a;
if m<b
then m:=b;
if m<c
then m:=c;
```

Упражнения:

1. Установите, сколько ветвей в данном алгоритме и постройте для каждой ветви тест. Подойдут ли для тестирования этого решения тесты предыдущего решения?

2. Постройте алгоритмы, аналогичные решению 2, выбрав в качестве начального значения для максимума переменные b и c. Сравните полученные решения между собой. В чем сходство? Различие? Выполните тесты решения 1 на построенных алгоритмах. Достаточно ли этих тестов? Предложите другие тесты для ваших алгоритмов.

3. Напишите алгоритм, формальная запись которого такова: m=max(max(a,b),max(a,c),max(b,c)). Постройте тесты для полученного алгоритма. Сравните это решение с предыдущими.

4. Напишите алгоритм для нахождения минимального из трех.

5. Установите, какую задачу решают алгоритмы, построенные по формальным записям: а) m=max(min(a,b),c); б) m=min(max(a,b),c). Постройте для них тесты.

6. Установите, какую задачу решает следующий фрагмент программы?

```
if a>b
then m:=a
else m:=b;
if m<c
then m:=c
```

Постройте для него тесты. Сравните это решение с предыдущими.

7. Установите, какую задачу решает следующий фрагмент программы?

```
if (a>b) and (a>c)
then      m:=a          { ветка 1 }
else      if b>c
          then m:=b     { ветка 2 }
          else m:=c     { ветка 3 }
```

Укажите, какая ветвь выполняется в каждом из заданных исходных соотношений переменных программы:

1) a < b < c,
2) a < c < b,
3) b < a < c,
4) b < c < a,
5) c < a < b.
6) c < b < a,
7) a = b = c,
8) a = b < c,
9) c < a = b,
10) a < b = c,
11) b = c < a,
12) a = c < b,
13) b < a = c.

Пример 9.26. Написать программу для нахождения среднего по величине значения из трех заданных.

Решение. Составим спецификацию. Дано: a,b,c - вещественные числа. Получить: n - среднее по величине из трех заданных.

На первый взгляд кажется, что алгоритмы, формальная запись которых приведена в упражнении 5 а и 5 б, решают поставленную задачу. Однако это неверно. Для того чтобы убедиться в этом, будем рассуждать так. Средним по величине может оказаться любое из трех, следовательно, имеется три возможности. Выбрав и зафиксировав среднее, мы имеем две возможности распределения максимального и минимального значений среди оставшихся. Поэтому количество возможных комбинаций исходных данных равно 3·2·1=6. Пусть для определенности значения исходных данных выбираются из множества { 1,2,3 }. Возможные наборы исходных данных приведены в табл. 9.1.

Таблица 9.1

Варианты исходных данных для поиска среднего из трех.

№ п/п	**a**	**b**	**c**	**Результаты работы алгоритма 5а**	**Результаты работы алгоритма 5б**	**Соотношения исходных данных**
1	1	2	3	3 неверно	2	a<b<c
2	1	3	2	2	2	a<c<b
3	2	1	3	3 неверно	2	b<a<c
4	2	3	1	2	1 неверно	c<a<b
5	3	1	2	2	1 неверно	b<c<a

6	3	2	1	2	2	c<b<a

Алгоритм 5 а дает неверные результаты в первом и третьем случаях, алгоритм 5 б - в четвертом и пятом случаях.

Для решения поставленной задачи можно воспользоваться одной из следующих формул: n = min(max(a,b),max(a,c),max(b,c)) или n = max(min(a,b),min(a,c),min(b,c)).

Для построения алгоритма можно использовать следующий прием. Построим таблицу, в которой перечислим все возможные условия, сравнивающие исходные данные. Такая таблица называется таблицей решений. В нашем случае их три: a>b, a>c, b>c. Каждая из этих проверок может быть истинной или ложной, следовательно, возможны 8 вариантов комбинаций значений истинности. Все варианты могут быть сведены в табл.9.2. В ней истина (true) обозначена 1, а ложь (false) - 0.

Таблица 9.2

Таблица решений при поиске среднего из трех

Номер варианта из табл. 9.1.	**6**	**5**	**•**	**3**	**4**	**•**	**2**	**1**
a>b	1	1	1	1	0	0	0	0
a>c	1	1	0	0	1	1	0	0
b>c	1	0	1	0	1	0	1	0
Результат: среднее по величине	**b**	**c**	**Проти-воре-чие**	**a**	**a**	**Проти-воре-чие**	**c**	**b**

Как видно из таблицы, возможных комбинаций значений истинности трех выбранных условий 8, а осмысленных (имеющих смысл) комбинаций 6. Следовательно, две комбинации значений истинности противоречивы. Эти комбинации в табл. 9.2 отмечены точками. Рассмотрим первую комбинацию, отмеченную точкой. Здесь a>b и неверно, что a>c, т.е. a<=c вместе дают истинное условие b<a<=c, что противоречит третьему условию b>c. Убедитесь самостоятельно в наличии второго противоречия. Противоречия показывают, что в комбинациях, отмеченных точками, и в соседних с ними комбинациях нет необходимости проверять последнее условие (b>c). Этот факт отражается в таблице прочерком. Преобразованная таблица представлена табл. 9.3 следующим образом

Таблица 9.3

Таблица решений с устраненными противоречиями

a>b	1	1	1	0	0	0
a>c	1	1	0	1	0	0
b>c	1	0	-	-	1	0
Результат	b	c	a	a	c	b

В преобразованной таблице решений 6 комбинаций, что соответствует 6 веткам алгоритма. Построим алгоритм по полученной таблице решений. Проверка первого условия разделяет таблицу на две. Схематически эту проверку можно представить так:

if a>b
then { 1 }

a>c	1	1	0
b>c	1	0	-
	b	c	a

else { 2 }

a>c	1	0	0
b>c	-	1	0
	a	c	b

Анализируя часть таблицы, попавшую в ветку { 1 }, выясняем, что проверка условия a>c вновь разбивает эту ветку на две, причем в случае невыполнения условия больше проверок не нужно, т.к. ответ получается сразу. Точно так же разбивается ветка { 2 }. В результате получаем:

```
if a>b
then    if a>c
        then { 11 }
```

b>c	1	0
	b	c

```
        else n:=a       { 12 }
else    if a>c
        then n:=a       { 21 }
        else { 22 }
```

b>c	1	0
	c	b

Ветки { 1 } и { 2 } разбились на две, образовав четыре новых ветки, причем ветки { 11 } и { 22 } еще содержат часть таблицы, которую нужно заменить условным оператором. В результате окончательно получаем:

```
if a>b
then    if a>c
        then    if b>c
                then n:=b
                else n:=c
        else n:=a
else    if a>c
        then n:=a
        else    if b>c
                then n:=c
                else n:=b.
```

Упражнения:

1. Сколько веток в последнем фрагменте программы?
2. Постройте тесты для каждой ветки.

3. Запишите фрагмент программы для присвоения переменной m1 минимального, переменной m2 среднего, переменной m3 максимального из трех целых чисел. Попробуйте найти несколько решений данной задачи.

9.4. Таблицы решений

Таблицей решений (ТР) является таблица, отображающая зависимость действий, подлежащих выполнению, от условий, при которых они могут выполняться.

Пример 9.27. ТР, поясняющая действия, которые нужно выполнить при переходе улицы, регулируемой светофором:

Какой сигнал светофора?	Красный	Желтый	Зеленый	Красный	Желтый	Зеленый
Нужно ли переходить улицу?	да	да	да	нет	нет	нет
Стой	**x**					
Приготовься		**x**				
Переходи			**x**			
Займись другим делом				**x**	**x**	**x**

Пример 9.28. ТР, поясняющая ситуацию, возникающую у витязя на развилке дорог в русских народных сказках: «Направо поедешь - себя спасать, коня потерять. Налево поедешь - коня спасать, себя потерять. Прямо поедешь - женату быть»:

Ехать направо	1		
Ехать налево		1	
Ехать прямо			1
Коня потерять	x		
Голову потерять		x	
Женату быть			x

Пример 9.29. ТР, поясняющая путь к профессиональному счастью: «Нет ничего хорошего, если не найдешь свой талант; хуже, найдя, полениться его отрыть; совсем плохо, отрыв, тут же зарыть обратно»:

Нашел свой талант?	1	0	1	1
Поленился отрыть?	0	-	1	0
Зарыл обратно?	0	-	-	1
Нет ничего хорошего		x		
Еще хуже			x	
Совсем плохо				x
Первый шаг к профессиональному счастью	x			

Пример 9.30. ТР представляет древнерусское стихотворение:
«Тот, кто не знает и не знает, что он не знает, - глупец, избегай его.

Тот, кто не знает и знает, что он не знает, может научиться, научи его.
Тот, кто знает и не знает, что он знает, спит, разбуди его.
Тот, кто знает и знает, что он знает, - пророк, учись у него.»

C1 Он знает	N	N	Y	Y
C2 Он знает C1	N	Y	N	Y
A1 Избегай его	x			
A2 Научи его		x		
A3 Разбуди его			x	
A4 Учись у него				x

Как видно из приведенных таблиц, ТР состоит из четырех частей: столбца условий, входов условий, столбца действий и входов действий. Взаимное размещение этих частей показано ниже:

Столбец условий	Вход условий
Столбец действий	Вход действий

Столбец условий содержит условия, проверки, вопросительные предложения, совокупность ответов на которые позволяет описывать ситуацию. Вход условий представляет собой перечень всех возможных ответов на указанные вопросы. Столбец действий содержит описания всех действий, которые могут быть выполнены в различных ситуациях. Вход действий содержит пометки, указывающие, какие действия необходимо выполнить при каждой комбинации результатов проверок условий. Совокупность входа условий и входа действий образует часть таблицы, называемую входом. Каждый столбец входа таблицы называется правилом. Различают таблицы решений с ограниченным и расширенным входами. Таблица решений с ограниченным входом в столбце условий содержит вопросы, на которые можно ответить «да»(1,Y) или «нет»(0,N), а каждая позиция входа действий содержит пометку, указывающую на необходимость применения данного метода, или остается незаполненной, если действие не применяется. Если правила заполнения части входов отличаются от описанных, тогда говорят, что таблица имеет расширенный вход. Таблица с расширенным входом приведена в примере 9.27. Таблицы с ограниченным входом приведены в примерах 9.28, 9.29, 9.30.

Таблица решений может быть полной и неполной. ТР называется полной, если в ней содержатся столбцы для всех комбинаций ответов «да» и «нет». Полные таблицы приведены в примерах 9.29 и 9.30. ТР примера 9.29 представлена в компактном виде. Здесь совмещаются некоторые столбцы друг с другом за счет объединения несущественных условий. Правила совмещаются только в том случае, если имеют один и тот же набор действий и различаются входными условиями. Несуществующие условия в каждом случае отмечаются прочерком. Каждый столбец, содержащий прочерки, заменяет несколько столбцов, количество которых определяется двойкой в степени, равной количеству прочерков. В самом деле, если в столбце один прочерк, то он заменяет два правила. В одном из них вместо прочерка стоит «Да»(1,Y), в другом - «Нет»(0,N). Еще один прочерк заменяет еще два правила, а всего 2·2 - четыре правила. В соответствии с правилом произведения комбинаторики в случае n прочерков имеем $2 \cdot 2 \cdot \ldots \cdot 2 = 2^n$. Количество правил,

представленных в ТР i-м столбцом, будем обозначать k_i. Полная ТР с ограниченным входом содержит 2^n правил, где n - количество условий.

Получить полную таблицу можно, если добавить к произвольной таблице правило ИНАЧЕ, которое будет принимать значение «истина», если все другие правила ТР будут ложными. ТР с правилом ИНАЧЕ приведена ниже:

На прогулку!

На улице сильный ветер? Идет дождь? Идет град? Небо затянуто тучами? ИНАЧЕ?	Да Да Да Да Да
Оденьтесь теплее Наденьте плащ и возьмите зонт Лучше подождать с прогулкой Прогулка может получиться весьма приятной	x x x x x

ТР избыточна, если хотя бы два ее столбца перекрывают друг друга по значениям условий и определяют одни и те же действия. Пример избыточной таблицы:

C1	1	-
C2	-	0
A1	x	x

Таблица решений противоречива, если хотя бы два столбца ее перекрывают друг друга по значениям условий, но определяют различные действия. Пример противоречивой ТР:

C1	1	-
C2	-	0
A1	x	
A2		x

Работа по ТР осуществляется следующим образом:

1) устанавливается истинность всех условий, используемых в ТР для заданных исходных данных;

2) выбирается правило ТР, содержащее те же значения истинности условий, что и установленные на первом шаге, при этом вместо прочерка можно поставить 1 или 0. Используя пометку во входе действий, в столбце действий считываем то, что нужно выполнить. Если таких правил несколько и каждое помечает одно и то же действие, то таблица избыточна и одно из правил можно отбросить. Если таких правил несколько и каждое помечает разные действия, то таблица противоречива и ей пользоваться нельзя.

Упражнения:

1. Какое решение подскажет ТР примера 9.27, если Вы хотите перейти улицу, но горит желтый сигнал светофора?

2. Что произойдет, если витязь из примера 9.28 поедет направо?

3. При выполнении каких условий в ТР примера 9.29 будет выбрано действие «еще хуже»?

4. В каких случаях следует учиться у кого-либо (ТР примера 9.30)?

5. Сколько правил должна содержать полная таблица решений с ограниченным входом, содержащая три условия?

6. Задана ТР:

B1	1	0	1	1
B2	0	-	0	1
B3	1	-	0	-
B4	1	-	-	-
M1	x			
M2		x		
M3			x	
M4				x

Ответьте на вопросы по этой таблице решений:

1) укажите, сколько правил заменяет каждый из столбцов ТР;

2) является ли эта ТР полной;

3) является ли эта ТР противоречивой;

4) могло ли быть в этой ТР больше четырех действий?

7. Задана ТР:

	R1	R2	R3	R4	ИНАЧЕ
B1=0	N	-	N	Y	
B2=0	-	Y	N	Y	
B3=0	Y	N	N	-	
Действия	M1	M2	M3	M4	M5

Ответьте на вопросы по этой ТР:

1) укажите, сколько правил заменяет каждый из столбцов;

2) является ли эта ТР полной;

3) какие действия будут выполнены, если исходные данные таковы:

а) B1=0, B2=0, B3=3;

б) B1=0, B2=2, B3=0;

4) что произойдет, если поменять местами столбцы R2 и R4, а исходные данные оставить такими же, как в задании 3);

5) является ли эта ТР противоречивой?

8. Построить ТР по стихотворению М.Смородинова «Сказка»:

«Налево, направо иль прямо?
Вот - думай, копье накреня.
Там - смерть,
там - лукавая дама,
А там - потеряешь коня.»

9. Подобрать стихотворения или отрывки из прозы, которые можно представить в виде таблиц решений.

10. Составить ТР, определяющую плату в плавательный бассейн, если приняты следующие расценки (цены условны). Дети до 14 лет - одна тысяча рублей. Подростки до 18 лет - полторы тысячи рублей. Взрослые - две тысячи рублей. Если

в группе не менее 10 человек, то расценка снижается на 500 рублей с каждого. Учащиеся школ, студенты, учащиеся ПТУ пропускаются по детскому тарифу.

9.4.1. Построение таблиц решений

Для построения ТР нет единого алгоритма, да и нет алгоритма вообще. Можно привести самые общие рекомендации:

1) выделить ситуации, указанные в задаче;

2) записать их в таблицу;

3) проверить таблицу на полноту.

Пример 9.31. Представить в виде ТР решение уравнения степени не выше второй: $ax^2+bx+c=0$.

Решение. Уравнение этого вида при a=0 становится линейным, а при a<>0 - квадратным. Квадратное уравнение имеет в зависимости от знака дискриминанта два или один действительный корень, или не имеет действительных корней. Линейное уравнение может иметь один корень при b<>0, не иметь корней при b=0 и c<>0, иметь бесконечное количество корней при b=0 и c=0.

Запишем приведенные рассуждения в виде ТР, где a,b,c - коэффициенты, d - дискриминант:

a=0	1	1	1	0	0	0	0
b=0	1	1	0	-	-	-	-
c=0	1	0	-	-	-	-	-
d>0	-	-	-	1	0	0	1
d=0	-	-	-	0	1	0	1
Линейное x=-b/c			x				
Линейное бесконеч.	x						
Линейное нет		x					
Квадратное x1,x2				x			
Квадратное x					x		
Квадратное нет						x	

невозможная комбинация условий ⇑

Проверим полноту этой таблицы. Первый столбец заменяет 4 правила($2^{\text{количество прочерков}}$), второй - 4 правила, третий - 8 правил, четвертый - 4 правила, пятый - 4 правила, шестой - 4 правила, седьмой столбец содержит невозможные комбинации условий, их всего 4. Таким образом, всего условий 32.

Всего условий должно быть $2^{\text{количество условий}}=2^5=32$, следовательно, ТР полна.

Пример 9.32. Построить ТР для решения системы двух линейных уравнений с двумя неизвестными:

$$\begin{cases} a_1 x + b_1 y = c_1, \\ a_2 x + b_2 y = c_2. \end{cases}$$

Решение. Вычислим: $\Delta=a_1b_2-a_2b_1$,

$\Delta_1=c_1b_2-c_2b_1$,

$\Delta_2=a_1c_2-a_2c_1$.

Δ=0	0	1	1	1	1

$\Delta_1=\Delta_2=0$	-	0	1	1	1
$a_1=b_1=a_2=b_2=0$	-	-	0	1	1
$c_1=c_2=0$	-	-	-	0	1
Одно решение	x				
Система несовместна,нет решений		x		x	
Система неопределенная, бесконечное количество решений			x		x

Упражнение. Проверьте полноту построенной ТР.

9.5. Задачи, приводящие к ветвящимся алгоритмам

Пример 9.33. Определить, поместится ли квадрат площади P в круг площади S или круг поместится в квадрат, или они не поместятся друг в друга.

Решение. Квадрат поместится в круг, если его диагональ будет меньше или равна диаметру круга. Круг поместится в квадрат, если его диаметр будет меньше стороны квадрата или равен ей. Если не выполняются оба указанных условия, то круг и квадрат не поместятся друг в друга.

Обозначим длину стороны квадрата - a, длину его диагонали - sqrt(2)*a, а длину радиуса круга - r. Тогда a=sqrt(P), а r=sqrt(S/π). Построим таблицу решений и перейдем от нее к программе:

$\sqrt{2P} \le 2\sqrt{\frac{S}{\pi}}$	1	0	0	1
$2\sqrt{\frac{S}{\pi}} \le \sqrt{P}$	0	1	0	1
Квадрат поместится в круг	x			
Круг поместится в квадрат		x		
Квадрат и круг не поместятся друг в друга			x	

невозможная комбинация условий ⇑

```
if sqrt(2*P)<=2*sqrt(S/pi)
then write('Квадрат поместится в круг')
else      if 2*sqrt(S/pi)<=sqrt(P)
          then write('Круг поместится в квадрат ')
          else write('Квадрат и круг не поместятся друг в друга ')
```

Пример 9.33. Любую целочисленную сумму денег, превышающую 7 рублей, можно выплатить без сдачи трешками и пятерками. По заданному целому положительному n определить пару целых неотрицательных чисел a и b, таких, что n=3·a+5·b.

Решение. Заметим, что если выплачивать деньги трешками, то в остатке могут получиться 0, 1 или 2. Если в остатке получился ноль, то вся сумма выдается трешками (a=n div 3, b=0). Если в остатке получился 1 рубль, нужно забрать обратно три трешки и выдать две пятерки (3*3+1=10 div 5 =2, a=n div 3-3, b=2). Это решение можно применять начиная с n=10. Если в остатке получилось два рубля, то нужно забрать одну трешку и выдать одну пятерку (3+2=5 div 5 =1, a=n div 3-1, b=1). Построим таблицу решений:

n mod 3=0 n mod 3=1 n mod 3=2	1 1 1
a=n div 3, b=0 a=n div 3-3, b=2 a=n div 3-1, b=1	x x x

От нее перейдем к программе:

```
ost:=n mod 3;
chas:=n div 3;
if ost=0
then write('3*',chas,'+5*0=',n)
else        if ost=1
            then write('3*',chas-3,'+5*2=',n)
            else write('3*',chas-1,'+5*1=',n).
```

Возможен вариант этого решения, когда деньги выдаются пятерками. В этом случае возможны остатки 0, 1, 2, 3 и 4 рубля. Если в остатке получается ноль, то (a=0, b=n div 5) все деньги можно выдать пятерками. Если в остатке получается один, то можно забрать одну пятерку и выдать две трешки (5+1=6 div 3=2, a=2, b=n div 5-1). Если в остатке получается два, то можно забрать две пятерки и выдать 4 трешки (2*5+2=12 div 3 =4, a=4, b=n div 5-2). Если в остатке получается три, то эту тройку можно выдать одной трешкой (a=1, b=n div 5). Если в остатке получается четыре, то можно забрать одну пятерку и выдать три трешки (5+4=9 div 3=3, a=3, b=n div 5-1). Построим таблицу решений:

n mod 5=0 n mod 5=1 n mod 5=2 n mod 5=3 n mod 5=4	1 1 1 1 1
a=0, b=n din 5 a=2, b=n div 5-1 a=4, b=n div 5-2 a=1, b=n div 5 a=3, b=n div 5-1	x x x x x

От нее перейдем к программе:

```
ost:=n mod 5;
chas:=n div 5;
if ost=0
then write('3*0+5*',chas,'=',n)
else        if ost=1
            then write('3*2+5*',chas,'=',n)
            else    if ost=2
                    then write('3*4+5*',chas,'=',n)
                    else    if ost=3
                    then write('3*1+5*',chas,'=',n)
                    else write('3*3+5*',chas,'=',n).
```

Пример 9.34. В старой лавке Аладин нашел пять волшебных ламп, заставить работать которые можно было, ударив их друг о друга. В момент удара из ламп сыпались искры, причем при любом ударе из каждой конкретной лампы высыпалось одно и то же количество искр, но не совпадающее с количеством искр, высыпавшихся из другой лампы. Волшебство появлялось из той лампы, искр из которой высыпалось больше. Написать программу для определения самой волшебной лампы.

Решение. Обозначим лампы именами L1, L2, L3, L4, L5. Переменные с этими именами будут хранить количество искр, высыпавшихся при ударе из одноименной лампы. Тогда задача сводится к поиску максимального значения из пяти заданных:

```
program pr1;
var  L1,L2,L3,L4,L5 :integer;{количество искр в соответствующей лампе}
     m : integer; {максимальное количество искр среди просмотренных ламп}
     n : integer; {номер лампы с максимальным количеством искр}
begin      write('введите пять чисел - количество искр, ',
                 'высыпавшихся из пяти ламп ');
           readln(L1,L2,L3,L4,L5);
           m := L1; n := 1; {пусть пока самой волшебной будет первая лампа}
           if m < L2
           then begin m := L2; n := 2 end;
           if m < L3
           then begin m := L3; n := 3 end;
           if m < L4
           then begin m := L4; n := 4 end;
           if m < L5
           then begin m := L5; n := 5 end;
           write ( 'Самая волшебная лампа ',n,'. В ней ', m, ' искр.')
end.
```

Пример 9.35. Решить систему уравнений

$$\begin{cases} a \cdot x = b, \\ x + c \cdot y = 1 \end{cases}$$

Решение. Если a=0 и b<>0, то x не существует и у системы нет решений. Если a=0 и b=0, то x может быть x=1-c·y, где y - любое. Если a<>0, то x=b/a. Подставляя во второе уравнение, получаем c·y=(a-b)/a. Если c=0 и a-b=0 или a=b, то y - любое, а x=1. Если c=0 и a-b<>0 или a<>b, то y не существует и система решений не имеет. Если c<>0, то y=(a-b)·c/a, а x=b/a.

Построим таблицу решений:

a=0, b<>0	1				
a=0, b=0		1			
a<>0, c=0, a=b			1		
a<>0, c=0, a<>b				1	
a<>0, c<>0, a<>b					1
решений нет	x			x	
x=1-c*y, y - любое		x			
x=1, y - любое			x		

x=b/a, y=(a-b)*c/a	x

От нее перейдем к программе.

```
if a=0
then    if b<>0
        then write('решений нет')
        else write('x=1-c*y, y - любое')
else    if c=0
        then    if a=b
                then write('x=1, y - любое')
                else write('нет решений')
        else write('x=',b/a,'y=',(a-b)*c/a).
```

Пример 9.36. Даны три числа. Проверить, могут ли они быть длинами сторон треугольника, и, если могут, определить вид треугольника: равносторонний, равнобедренный или разносторонний.

Решение. Три числа могут быть длинами сторон треугольника, если сумма любых двух больше третьего числа. Треугольник равносторонний, если все его стороны равны. В равнобедренном треугольнике равны две стороны. У разностороннего треугольника все стороны разные. Обозначим исходные данные буквами a, b и c. Построим таблицу решений:

a+b>c	0			1	1	1	1	1
a+c>b		0		1	1	1	1	1
b+c>a			0	1	1	1	1	1
a=b				1	1	0	0	0
a=c				1	0	-	1	0
b=c				1	-	1	0	0
не треугольник	x	x	x					
треугольник				x	x	x	x	x
равносторонний				x				
равнобедренный					x	x	x	
разносторонний								x

От нее перейдем к программе:

```
if (a+b>c) and (a+c>b) and (b+c>a)
then    begin  write('треугольник ');
               if a=b
               then   if a=c
                      then write('равносторонний')
                      else write('равнобедренный')
               else   if b=c
                      then write('равнобедренный')
                      else   if a=c
                             then write('равнобедренный')
                             else write('разносторонний')
        end
else write('не треугольник').
```

Пример 9.37. Принадлежит ли точка M(x,y) кольцу с центром в точке O(2,3), внешним радиусом 6 см и внутренним радиусом 4 см.

Решение. Точка принадлежит кольцу, если она расположена от центра на расстоянии большем, чем внутренний радиус, и меньшем, чем внешний радиус. Отсюда $16 \le (x-2)^2+(y-3)^2 \le 36$. Запись на Паскале приведена ниже:

```
if (16<=sqr(x-2)+sqr(y-3)) and (sqr(x-2)+sqr(y-3)<=36)
then write('точка принадлежит кольцу')
else write('точка НЕ принадлежит кольцу').
```

Пример 9.38. Задана функция

$$y = \begin{cases} x+1, & \text{если } x < 5, \\ x^2 - 2, & \text{если } -5 \le x \le 5 \\ x^3 + 3, & \text{если } x \ge 5. \end{cases}$$

Написать фрагмент программы на Паскале для вычисления значения этой функции.

Решение. Функция определена на всей числовой оси. Точки -5 и 5 разбивают её на три интервала (рис.9.1). В зависимости от того, в какой интервал попадает аргумент x, функция y вычисляется по соответствующей формуле. При построении программы учтём, что если ложно условие x<-5, то в соответствии с логическими законами автоматически истинно x≥-5, поэтому выполнение неравенства x≥-5 проверять не нужно.

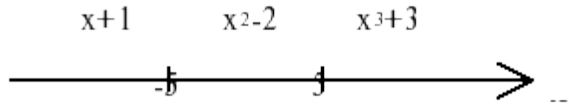


Таблица решений для задачи имеет вид:

x<-5	1	0	0
x<5	-	1	0
y=x+1	x		
$y=x^2-2$		x	
$y=x^3+3$			x

Фрагмент программы, построенный по таблице решений:

```
if x<-5
then y:=x+1
else      if  x<5
          then y:=x*x-2
          else y:=x*sqr(x)+3.
```

Упражнения: 1. Постройте график функции задачи 9.38.

2. По таблице решений напишите фрагмент программы, вычисляющей значение функции. Постройте график этой функции.

x<-2	1	0	0	0	0
x<0	-	1	0	0	0
x<2	-	-	1	0	0
x<2.5	-	-	-	1	0
x^2	x				

$2x^2-3x$	x			
$2x^2$		x		
$2x^2-3x+4$			x	
0				x

3. Пусть $z=6x^6-5x^5+4x^4-3x^3+2x^2-x+7$. Напишите фрагмент программы на Паскале, вычисляющей значение функции, постройте ее график.

$$y = \begin{cases} |z+2|, & \text{æ}^{\text{o}} \text{Ł } z < 0, \\ 2|z|, & \text{æ}^{\text{o}} \text{Ł } 0 \le z < 1, \\ |2z+2|, & \text{æ}^{\text{o}} \text{Ł } 1 \le z < 3, \\ 2|z|+2, & \text{æ}^{\text{o}} \text{Ł } 3 \le z < 5 \\ 2|z|-2, & \text{æ}^{\text{o}} \text{Ł } z \ge 5. \end{cases}$$

4. По фрагментам Паскаль-программ восстановите формулу, вычисляющую значение каждой функции, постройте таблицы решений и графики функций:

а)
```
if  x<2
then y:=x
else    if x<3
        then y:=2
        else y:=x-1;
```

б)
```
if x<-1
then y:=1/x
else    if x<=2
        then y:=sqr(x)
        else y:=4;
```

в)
```
if x<-0.5
then y:=1/abs(x)
else    if x<1
        then y:=2
        else y:=1/(x-0.5);
```

г)
```
if abs(x)>2
then y:=aqr(x)
else if x<0
      then y:=-2*x
      else  if x>=1
            then y:=4
            else y:=4*sqr(x);
```

д)
```
    if sqr(x)>2
    then    if x>2
            then y:=sqr(x)*x
            else y:=8
    else y:=8*sqr(x).
```

Пример 9.39. Периодическая функция f(x) определена на всей числовой оси и имеет период 2. График этой функции на отрезке [0;2] приведён на рис.9.2. Написать фрагмент программы, вычисляющей значения y по значению x.

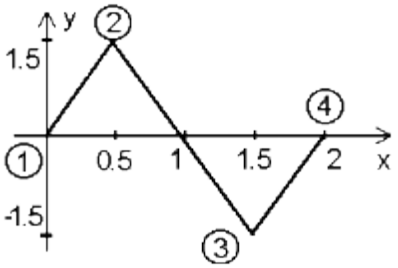


Рис. 9.2

Решение. Сначала установим уравнения прямых, составляющих график заданной функции. График состоит из трех отрезков прямых:

1) отрезок между точками 1(0;0) и 2(0.5;1.5);
2) отрезок между точками 2(0.5;1.5) и 3(1.5;-1.5);
3) отрезок между точками 3(1.5;-1.5) и 4(2;0).

Будем искать уравнения прямых в виде y=ax+b. (*)

Для получения уравнения первой прямой подставим координаты двух известных её точек в уравнение (*) и решим полученную систему двух линейных уравнений с двумя неизвестными.

$$\begin{cases} 0 = 0 \cdot a + b, \\ 1.5 = 0.5 \cdot a + b \end{cases}$$

Отсюда, b=0, a=1.5/0.5=3. Уравнение первой прямой имеет вид y=3x.

Аналогичным образом поступаем при поиске уравнения второй прямой.

$$\begin{cases} 1.5 = 0.5 \cdot a + b, \\ -1.5 = 1.5 \cdot a + b \end{cases}$$

Отсюда, b=1.5-0.5a;
-1.5=1.5a+1.5-0.5a;
a=-3; b=3.

Уравнение второй прямой имеет вид y=-3x+3.

Для третьей прямой имеем

$$\begin{cases} -1.5 = 1.5 \cdot a + b \\ 0 = 2 \cdot a + b. \end{cases}$$

Отсюда, b=-2a;
-1.5=1.5a-2a;
a=3; b=-6.

Уравнение третьей прямой имеет вид y=3x-6.

Таким образом в пределах отрезка [0;2] (один период) значение функции можно вычислить по формуле

$$y = \begin{cases} 3x, & \text{æ}^{\text{o}} \text{Ł } 0 \le x < 0.5, \\ -3x + 3, & \text{æ}^{\text{o}} \text{Ł } 0.5 \le x < 1.5 \\ 3x - 6, & \text{æ}^{\text{o}} \text{Ł } 1.5 \le x < 2. \end{cases}$$

Чтобы найти значение функции нужно отбросить целое число периодов так, чтобы оставшееся значение попало на отрезок [0;2]. Теперь, чтобы найти y, нужно решить задачу, похожую на предыдущую.

Для сопоставления точке x точки отрезка [0;2], воспользуемся формулой x-trunc(x/2)·2. Здесь trunc(x/2) - целое число периодов, помещающееся в x; trunc(x/2)·2 - длина такого количества периодов.

Следовательно, Паскаль-программа может быть такой:

```
x:=x-trunc(x/2)*2;
if x<0.5
then y:=3*x
else    if x<1.5
        then y:=-3*x+3
        else  y:=3*x-6
```

Упражнения:

1. Выполните трассировку предыдущего фрагмента для x=2.2; 7.4; 9.9; -2.2; -7.4; -9.9; 0; 1; 0.75; 1.8.

2. Постройте Паскаль-программу для вычисления значений периодической функции, определённой на всей числовой оси, заданной рис.9.3.

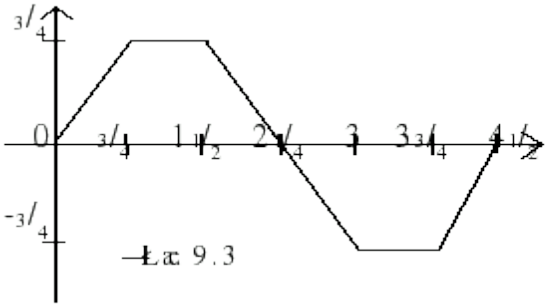
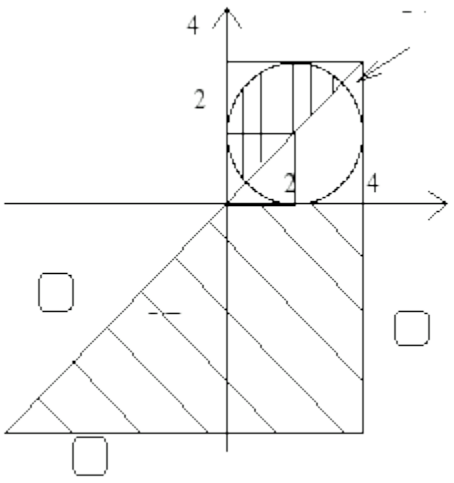


Рис. 9.3

Пример 9.40. Проверить, принадлежит ли точка M(x,y) заштрихованной на рис. 9.4 области.

Решение.

Область состоит из трёх подобластей: A, B и C. Точка будет принадлежать заштрихованной области, если она попадет в одну из трех подобластей, таким образом будет выполнено условие: A or B or C. Область A ограничена прямыми 1, 2, 3 и осью X. Уравнения прямых можно получить методом, описанным в предыдущей задаче.



Уравнение прямой 1: y= x.
Уравнение прямой 2: x= 4.
Уравнение прямой 3: y= -8.

Точка принадлежит области A, если она лежит ниже оси X ($y \le 0$),если она лежит выше прямой $y = -8$ ($y \ge -8$), если она лежит ниже прямой $y = x$ ($y < x$), если она лежит левее прямой $x=4$ ($x \le 4$).

На Паскале это запишется так: (y <= 0) and (y >= -8) and (y < x) and (x <= 4)

Точка принадлежит области B, если она лежит внутри окружности с центром в точке (2;2) и радиусом 2 и если она лежит выше прямой 1 ($y \geq x$). Уравнения окружности с центром в точке $x_ц$, $y_ц$ можно записать как
$(x-x_ц)^2+(y-y_ц)^2=R^2$.

На Паскале принадлежность точки области B запишется так:

(sqr(x-2)+sqr(y-2)<=4) and (y>=x).

Точка принадлежит области C, если она лежит ниже прямой 1 ($y \leq x$), если она лежит вне окружности $sqr(x-2)+sqr(y-2) \leq 4$, если она лежит левее прямой 2 ($y \leq 4$), если она лежит выше прямой $y=2$ ($y \geq 2$).

На Паскале это условие запишется так:

(y<=) and (sqr(x-2)+sqr(y-2)>=4) and (y<=4) and (y>=2).

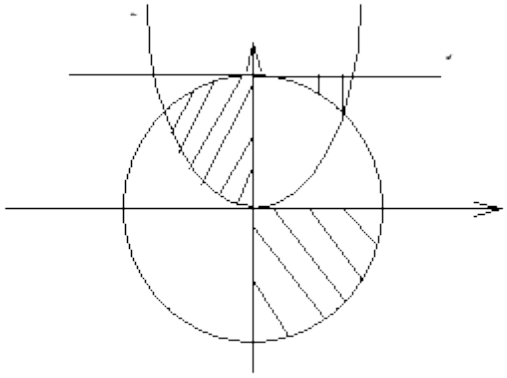
Окончательно получаем фрагмент программы:

```
if  (y<=0) and (y>=-8) and (y<=x) and (x<=4) or
    (sqr(x-2)+sqr(y-2)<=4) and (y>=x) or
    (y<=x) and (sqr(x-2)+sqr(y-2)>=4) and (y<=4 and (y>=2)
then write('Точка принадлежит указанной области')
else write('Точка Не принадлежит указанной области').
```

Упражнения:

1. Что изменится в решении, если в условии принадлежности точки области C опустить условие y>=2 ?

2. Проверить лежит ли точка M(x,y) в области, заштрихованной на рис. 9.5.



3. Проверить, пройдёт ли кирпич размерами a·b·c:

а) в прямоугольное окно размерами p·q;

б) в треугольное равностороннее окно с длиной стороны S;

в) в круглое окно радиуса R.

10. ЦИКЛЫ

10.1. Основные понятия. Виды циклов. Общий подход к построению циклов

Циклическим (циклом) называется такой алгоритм, в котором некоторая группа действий повторяется неоднократно. Группа действий, повторяемая в цикле, называется его *телом*. Однократное выполнение тела цикла называется *шагом*. Для того, чтобы алгоритм не зациклился (не стал бесконечным), циклом надо управлять. Для этого используется специальная величина - параметр цикла. *Параметр (переменная) цикла* - это такая величина, которая изменяется от шага к шагу и по значению которой определяется, продолжать исполнение цикла или его закончить.

Рассмотрим примеры тел циклических алгоритмов:

Пример 10.1. Заполнять бочку водой до тех пор, пока она не заполнится.

Решение. Здесь повторяются действия: налить воду в ведро; перелить воду из ведра в бочку, пока вода не достигнет ее краев.

Пример 10.2. Аудиторию готовят к покраске. Нужно выносить парты из нее, пока она не опустеет.

Решение. Повторяются действия: если аудитория не пуста, то вынести парту.

Пример 10.3. Положить в коробку десять конфет.

Решение. Повторяются действия: взять одну конфету; положить конфету в коробку; сосчитать положенную конфету.

Пример 10.4.Пройти десять километров.

Решение. Повторяются действия: пройти один километр; сосчитать уже пройденные километры.

Пример 10.5.Найти в книге нужный абзац.

Решение. Повторяются действия: открыть очередную страницу книги; просматривать страницу если нужный абзац найден, то завершить работу с результатом «НАЙДЕНО»; если книга закончилась, то завершить работу с результатом «НЕ НАЙДЕНО».

Пример 10.6. Выполнить домашнее задание.

Решение. Повторяются действия: выполнять домашнее задание; если надоело, то отложить выполнение; если пришли подруги, то отложить выполнение; если возникла другая уважительная или не очень уважительная причина,то отложить выполнение; если все выполнено, то закончить.

Приведенные примеры показывают, что циклы различаются способами окончания. В примере 10.1 заранее неизвестно, сколько ведер потребуется для заполнения бочки, поэтому приходится при переливании постоянно следить, не переливается ли вода через край. Назовем такие циклы итерационными. К этим циклам относится и пример 10.2.

В примере 10.3 заранее известно сколько конфет нужно положить в коробку, поэтому для определения окончания работы достаточно считать количество положенных конфет. Назовем такие циклы арифметическими. К ним относится и пример 10.4. Выполнение примера 10.2 можно также организовать как арифметический цикл, если заранее подсчитать, сколько парт стояло в аудитории.

В примере 10.5 цикл может завершиться по двум причинам: 1) если обнаружен искомый абзац; 2) если просмотрена вся книга. Назовем такие циклы *поисковыми*. К ним относится и пример 10.6, где количество причин завершения цикла больше двух.

В общем случае структура циклов может быть представлена одним из двух способов.

Первый способ организации цикла (цикл с предусловием):

1. Установка начальных значений переменных, использующихся или изменяющихся в цикле.
2. П Р О В Е Р К А: продолжить цикл?
   2 ДА - выполняется тело цикла;
   изменяется значение переменной цикла;
   возвращаемся на проверку в пункт 2.
   2 НЕТ - выходим из цикла на пункт 3.
3. Операторы, следующие за циклом.

Второй способ организации цикла (цикл с постусловием):

1. Установка начальных значений переменных, использующихся или изменяющихся в цикле.
2. Выполнение тела цикла.
   Изменение переменной цикла.
3. П Р О В Е Р К А: закончить цикл?
   3 ДА выход из цикла на пункт 4.
   3 НЕТ возвращаемся на пункт 2.
4. Операторы, следующие за телом цикла.

В первом способе организации цикла проверка продолжения цикла осуществляется до выполнения тела цикла. Если она дает ответ НЕТ при первом вхождении в цикл, то цикл может не выполниться ни разу. Организованный таким способом цикл называется циклом с предусловием.

При втором способе тело цикла всегда выполнится хотя бы один раз, потому что проверка окончания цикла осуществляется после выполнения тела. Такой цикл называется циклом с постусловием.

Для правильной организации цикла необходимо выполнить следующие шаги. Сначала следует установить действия, которые повторяются в цикле, т.е. его тело.

Затем следует попытаться ответить на вопрос: «Известно ли заранее сколько раз повторяется тело цикла?». Если на этот вопрос можно ответить утвердительно, то цикл является арифметическим и его параметром служит счетчик. Если ответ на предыдущий вопрос отрицательный, то отвечаем на следующий вопрос: «Сколько причин окончания цикла имеется?». Если имеется только одна причина окончания, то цикл итерационный. В качестве параметра здесь чаще всего выбирается величина, сравнимая с заданной точностью вычислений или преобразований. Если существует больше одной причины окончания цикла, то он поисковый. У такого цикла условие окончания задается сложным логическим выражением, содержащим конъюнкцию или дизъюнкцию.

На последнем шаге организации цикла необходимо правильно установить начальные значения для всех переменных, которые используются или изменяются в цикле. Эти переменные, как правило, находятся в условиях или в правых частях операторов присваивания. Для этого можно провести трассировку первоначального входа в цикл и определить, какие значения должны быть у используемых или изменяемых переменных.

10.2. Запись циклов на Паскале

В алгоритмическом языке Паскаль циклы могут быть записаны с помощью следующих операторов.

Цикл с предусловием:

while логическое_выражение do оператор.

Здесь оператор выполняется до тех пор, пока логическое выражение истинно. Для того чтобы цикл завершился, оператор в теле цикла должен в некоторый момент изменить значение логического условия. Оператор может быть любым оператором Паскаля, в том числе оператором цикла. В последнем случае цикл называется *вложенным*. Если тело цикла содержит больше одного оператора, то он оформляется как составной оператор.

Цикл с постусловием:

repeat операторы_тела_цикла until логическое_выражение.

Здесь операторы тела цикла выполняются до тех пор, пока логическое условие ложно. В теле цикла можно указывать несколько любых операторов Паскаля без дополнительных операторных скобок. Для выхода из цикла необходимо, чтобы операторы тела изменили значение логического выражения на истинное. Тело цикла этого оператора выполняется хотя бы один раз.

Цикл с параметром (вариант 1):

for параметр:=выражение_1 to выражение_2 do оператор.

В этом операторе выражение 1 и выражение 2 вычисляются только один раз при входе в цикл и в процессе выполнения не изменяются. Здесь в начале исполнения параметр получает значение выражения 1. Если значение параметра меньше или равно выражению 2, то выполняется оператор тела цикла. После выполнения тела цикла параметр автоматически получает значение, следующее за текущим значением параметра в соответствии с его типом. (Если параметр целый, то значение параметра увеличивается на единицу. Если параметр символьный, то выбирается следующий символ кодовой таблицы. Если логический, то выбирается следующее логическое значение. В любом случае параметр должен быть порядкового типа. Отсюда следует, что данные вещественного типа не могут быть параметрами цикла.) Если значение выражения 1 больше значения выражения 2, то цикл не выполнится ни разу. Параметр цикла в его теле изменять нельзя.

Цикл с параметром (вариант 2):

for параметр:=выражение_1 downto выражение_2 do оператор.

В этом операторе также выражение 1 и выражение 2 вычисляются только один раз при входе в цикл. Здесь в начале исполнения параметр получает значение выражения 1. Если значение параметра больше или равно выражению 2, то выполняется оператор тела цикла. После выполнения тела цикла параметр

автоматически получает предыдущее значение в соответствии с его типом. (Если параметр целый, то значение параметра уменьшается на единицу. Если параметр символьный, то выбирается предыдущий символ кодовой таблицы. Если логический, то выбирается предыдущее логическое значение. В любом случае параметр должен быть порядкового типа. Отсюда следует, что данные вещественного типа не могут быть параметрами цикла). Если значение выражения 1 меньше значения выражения 2, то цикл не выполнится ни разу. Параметр цикла в его теле изменять нельзя.

10.3. Примеры программ с циклами

10.3.1. Арифметические циклы

Для арифметического цикла известно заранее количество раз выполнения тела цикла, поэтому в качестве переменной используется счетчик.

Счетчик предназначен для выполнения N отсчетов. Для его организации необходимо знать начальное значение счетчика, при котором начинается отсчет, конечное значение, при котором завершается отсчет, шаг отсчетов, способ снятия отсчета (по строгому или нестрогому неравенству). Например, для N отсчетов подойдут следующие значения:

Начало	**Снятие отсчета**	**Конец**	**Шаг**
1	<=	N	+1
1	<	N+1	+1
0	<=	N-1	+1
0	<	N	+1
N	>	0	-1
N	>=	1	-1
N-1	>=	0	-1
N-1	>	-1	-1
1	<	2*N	+2 и т.д.

Часто при программировании бывают известны начальное значение счетчика, шаг и способ снятия отсчета. Для определения конечного значения при положительном шаге можно предложить следующие формулы.

Пусть н - начало отсчета, к - конец отсчета, ш - шаг отсчета, N - количество отсчетов. Тогда для нестрогого неравенства (<=) можно записать (к-н+ш)/ш = N.

После преобразований получаем: к-н+ш = N·ш,

к = N·ш+н-ш = (N-1) ·ш+н.

Например, для н=4, ш=3 и N=5 получаем 4·3+4=16. Проверка окончания: значение счетчика <= 16. Цикл выполнится для значений счетчика: 4, 7, 10, 13, 16.

Для строгого неравенства можно записать (<) (к-н)/ш = N. После преобразований получаем: к-н = N·ш, к = N·ш+н.

Например, для н=4, ш=3 и N=5 получаем 5·3+4=19. Проверка окончания: значение счетчика < 19. Цикл выполнится для значений счетчика: 4, 7, 10, 13, 16.

Предлагаем самостоятельно вывести формулы для отрицательного шага.

Пример 10.7. Написать программу для вычисления суммы N первых натуральных чисел.

Решение. Для решения задачи необходимо вычислить: 1+2+3+...+N. Понятно, что для нахождения суммы нужно каждый раз прибавлять очередное слагаемое; таким образом, для решения задачи необходимо организовать цикл. Исходным данным для решения является N.

Выясним, что повторяется в теле цикла. Для этого попробуем вычислять сумму вручную. Начинаем суммировать с нуля. Тогда

$s_0 = 0$,
$s_1 = 0+1$,
$s_2 = 0+1+2$,
$s_3 = 0+1+2+3$.

Найдем, чем каждый шаг отличается от соседнего. Для этого подсчитаем разности: $s_1 - s_0 = 0+1 - 0 = 1$,
$s_2 - s_1 = 0+1+2 - (0+1) = 2$,
$s_3 - s_2 = 0+1+2+3 - (0+1+2) = 3$.

Отсюда $s_1 = s_0 + 1$,
$s_2 = s_1 + 2$,
$s_3 = s_2 + 3$.

Рассуждая аналогично, можем заключить, что в общем случае для нахождения следующего значения суммы нужно к ее предыдущему значению прибавить очередное слагаемое. Обозначим очередное слагаемое через i, а номер шага - через j. Тогда $s_j=s_{j-1}+i$ повторяется в цикле. Здесь индексы указывают, что это одна и та же величина (сумма), но в разные моменты времени. При программировании эти индексы указывать не нужно, т.к. оператор присваивания выполняется в два этапа. На первом этапе выбираются значения всех указанных справа переменных, вычисляется значение выражения, стоящего справа. Затем вычисленное значение пересылается в переменную, указанную слева. Формулы, в которых новое значение вычисляется на основе одного или нескольких старых значений, называются рекуррентными. Полученная нами формула является рекуррентной формулой первого порядка, т.к. для вычисления нового значения достаточно одного предыдущего значения.

Известно, что для вычисления всей суммы необходимо выполнить полученную формулу N раз. Следовательно, необходимо организовать арифметический цикл с параметром-счетчиком, обеспечивающим N отсчетов. Приводим далее программу на Паскале:

```
program pr1;
{ вычисление суммы натуральных чисел от 1 до N - вариант 1 }
var     N:integer;      { последнее число суммы - исходное данное }
        s:integer;      { сумма - результат }
        i:integer;      { очередное слагаемое }
        j:integer;      { счетчик цикла }
begin   write('Введите последнее число суммы ');
        readln(N);      { Ввод исходного данного }
        s:=0;           { Начинаем суммирование с нуля }
        i:=1;           { Первое слагаемое известно заранее }
        j:=0;           { Начальное значение счетчика }
        while j<N do { Нужно выполнить цикл N раз }
```

```
        begin  s:=s+i; {Добавили слагаемое к сумме }
               i:=i+1; {Перешли к следующему слагаемому }
               j:=j+1; { Увеличили счетчик на единицу }
        end;           { Закончилось тело цикла }
        write('Сумма чисел от 1 до ',N,' равна ',s)
                       { Вывод результата }
end.
```

Анализируя программу, можно заметить, что переменные i и j изменяются синхронно (т.е. по значению i можно установить значение j и наоборот), поэтому в данном случае переменная, изображающая слагаемое, может играть и роль счетчика. Тогда программа примет вид:

```
program pr2;
    { вычисление суммы натуральных чисел от 1 до N - вариант 2 }
    var    N:integer;    { последнее число суммы - исходное данное }
           s:integer;           { сумма - результат }
           i:integer;           { счетчик цикла и слагаемое }
    begin  write('Введите последнее число суммы ');
           readln(N);    { Ввод исходного данного }
           s:=0;         { Начинаем суммирование с нуля }
           i:=1;         { Начальное значение счетчика и слагаемого }
           while i<=N do       { Нужно выполнить цикл N раз }
           begin  s:=s+i;{ Добавили слагаемое к сумме }
                  i:=i+1;{ Увеличили счетчик и слагаемое на единицу }
           end;                { Закончилось тело цикла }
           write('Сумма чисел от 1 до ',N,' равна ',s)
                  { Вывод результата }
    end.
```

В этом вырожденном случае цикл можно отнести к итерационным. Он выполняется до тех пор, пока слагаемое не станет равным N. Можно считать этот цикл и арифметическим, учитывая двойной смысл параметра. После любого шага i цикла переменная s хранит сумму предыдущих i слагаемых.

После написания текста программы ее необходимо протестировать. Прежде убедимся, что цикл завершится. Это действительно так в случае, если $N \geq 1$, так как переменная i на каждом шаге цикла увеличивается от 1 до N. Если N - неположительно, то цикл не будет выполнен ни разу и будет выдан ответ 0, что соответствует нашим ожиданиям от правильно работающей программы.

Теперь необходимо проверить работу программы для такого N, для которого ответ заранее известен. Рассматриваемая программа предназначена только для иллюстрации методов построения алгоритмов, результат ее работы просто получить воспользовавшись формулой $s = N \cdot (N+1)/2$. Например, для N=10 по формуле получается ответ s=55. Выполнив программу для N=10, получаем тот же ответ.

Заметим далее, что программа позволит ввести только целое значение N, отвергнув данные других типов, например вещественные, строковые и т.д.

Теперь попробуем проверить работу программы для N=300. С удивлением обнаруживаем, что компьютер выдал ответ: s=-20386. Почему было получено

отрицательное число, ведь складывались положительные числа? Попробуем выполнить программу для N=200. Получим ответ 20100. Попробуем выполнить программу для N=400. Получим ответ 14662. Почему сумма для меньшего значения N превосходит сумму для большего значения N? Оказывается, все дело в переполнении. Значения суммы для N=300 и N=400 больше максимально допустимого значения для типа integer (здесь maxint=32767). Поэтому для этих значений программу использовать нельзя. Установим максимальное значение N, для которого программа еще может выдать правильный ответ. Для этого найдем решения неравенства $N\cdot(N+1)/2\leq 32767$. Данное неравенство сводится к квадратному: $N^2+N-65534\leq 0$. Решением этого неравенства будет интервал $-256.5\leq N\leq 255.5$. Следовательно, разработанная программа будет верна для всех N от 1 до 255. Если необходимо получать суммы для больших значений N, то необходимо перейти к другим типам данных, например longint или real, и внести соответствующие исправления в программу.

Установив границы применимости программы, можно закончить тестирование.

Заметим, что тот же цикл можно было записать с помощью оператора for:

```
s:=0;                          или    s:=0;
for i:=1 to N do s:=s+1;       for i:=N downto 1 do s:=s+i;
```

Записи в этом случае получились более короткими, потому что оператор for специально предназначен для записи арифметических циклов. Запись с помощью цикла repeat until может быть такой:

```
s:=0;
i:=1;
repeat    s:=s+i;
          i:=i+1
until i>N
```

Упражнения (Во всех упражнениях n - натуральное число):

1. Выполните трассировку программы PR2.

2. Выполните трассировку приведенного ниже фрагмента прграммы и ответьте на вопрос: «Какую задачу он решает?»

```
s:=0;
i:=n;
while i>0 do
begin     s:=s+i;
          i:=i-1
end;
```

Запишите данный фрагмент с помощью операторов for и repeat until.

3. Выполните трассировку приведенного ниже фрагмента прграммы и ответьте на вопросы:

1) Какую задачу решает данный фрагмент?

2) В каких случаях выполнится условный оператор, записанный после цикла?

```
s:=0;
i:=1;
j:=n;
while i<=j do
begin     s:=s+i+j;
```

```
            i:=i+1;
            j:=j-1
end;
if n mod 2 =1
then s:=s-n mod 2+1;
```

4. Проведите трассировку приведенного ниже фрагмента прграммы и ответьте на вопрос: «Какую задачу он решает?». Можно ли здесь переменную j выразить через переменную i и убрать переменную j из программы?

```
s:=0;
i:=56;
j:=1;
while i<=56+2*(n-1) do
begin      s:=s+j;
            j:=j+1;
            i:=i+2
end;
```

5. Напишите фрагменты программ, вычисляющих суммы:

а) $s = 1/2+1/3+1/4+...+1/n$,

б) $s = 1^2+2^2+3^2+...+n^2$,

в) $s = 1/2+2/3+3/4+...+n/(n+1)$.

10.3.2. Итерационные циклы

Пример 10.8. Написать программу для нахождения наибольшего общего делителя двух натуральных чисел.

Решение. Используя идею Евклида, будем вычитать из большего числа меньшее до тех пор, пока числа не сравняются. Получаем цикл, для которого неизвестно заранее количество раз выполнения, но причина окончания у него одна - числа сравнялись. Цикл этот завершится всегда, потому что уменьшаются натуральные числа, разность между двумя соседними натуральными числами и наименьшее натуральное число - единица. Такой цикл называется итерационным. Ниже приводится программа, реализующая решение задачи на Паскале:

```
program pr3;
{ вычисление НОД двух натуральных чисел }
var        a,b:integer;    { исходные натуральные числа }
           x,y:integer;    { переменные, используемые для поиска НОД }
begin      write('Введите два натуральных числа ');
           readln(a,b);
           x:=a; y:=b;              { A }
           while x<>y do            { B }
                 if x>y             { C }
                 then x:=x-y        { D }
                 else y:=y-x;       { E }
           write(x);                { F }
end.
```

Проведем тестирование программы. Для этого построим столько тестов, чтобы можно было пройти по каждой ветви программы хотя бы по разу. В приведенном фрагменте имеется пять ветвей:

первая ветвь - A - B;
вторая ветвь - B - C;
третья ветвь - B - F;
четвертая ветвь - C - D - B;
пятая ветвь - C - E - B.

Путей здесь бесконечно много. Примерами путей являются последовательности из следующих ветвей:

первый путь ветви - 1 - 3;
второй путь ветви - 1 - 2 - 4 - 3;
третий путь ветви - 1 - 2 - 5 - 3;
четвертый путь ветви - 1 - 2 - 4 - 2 - 5 - 3;
пятый путь ветви - 1 - 2 - 4 - 2 - 4 - 2 - 4 - 3;
шестой путь ветви - 1 - 2 - 5 - 2 - 5 - 2 - 4 - 3 и т. д.

Среди указанных путей есть такие, в которых используются не все ветви (это пути 1, 2, 3, 5), но есть и такие, в которых проходится каждая ветвь программы хотя бы по одному разу (это пути 4, 6).

Для построения теста выбранного пути построим сначала предикат пути, который будет принимать значение истинно, если в процессе исполнения программы реализуется данный путь, и ложно - при реализации других путей.

Для построения предиката пути проходим путь в обратном направлении и выписываем конъюнкцию всех условий, которые встречаются на этом пути. Если при обратном проходе пути осуществляется переход через оператор присваивания (переменная := выражение), то в предикат пути вместо всех вхождений переменной из левой части оператора присваивания подставляется выражение из его правой части. Предикаты пути, построенные так, как описано выше, всегда выражаются в терминах констант и входных переменных. Чтобы пройти заданный путь во время исполнения программы, необходимо только найти такие значения исходных данных, которые делают предикат пути истинным.

Рассмотрим процесс построения предиката для четвертого пути приведенной выше программы.

Вначале проходим ветвь 3 от F к B. При исполнении программы в блок F попадаем в случае ложности условия B, поэтому предикат этой ветви имеет вид P: X=Y.

Движемся дальше от B к C через E. При переходе оператора присваивания Y:=Y-X в предикате P все вхождения переменной Y заменим на выражение Y-X. Получаем P: X=Y-X или P: 2*X=Y. Поскольку E выполняется в случае ложности условия C, присоединяем его отрицание через конъюнкцию к P. P: (2*X=Y) and (X<=Y).

Движемся от C к B и присоединяем к P через конъюнкцию условие B (здесь оно должно быть истинным). P: (2*X=Y) and (X<=Y) and (X<>Y). В полученном предикате из первого члена конъюнкции 2*X=Y следует третий: X, не равный Y, поэтому третий член конъюнкции можно отбросить. Получим P: (2*X=Y) and (X<=Y).

Движемся от B к C через D. При переходе через оператор присваивания (X:=X-Y) все вхождения X заменяем выражением X-Y и присоединяем через конъюнкцию условие C (здесь оно истинно). P: (2*(X-Y)=Y) and (X-Y<=Y) или (2*X=3*Y) and (X<=2*Y).

Присоединяем условие C (здесь оно истинно). P: (2*X=3*y) and (x<=2*y) and (x<y). Анализируя полученный предикат, замечаем, что если истинен предпоследний член конъюнкции, то истинен и последний. Таким образом, последний член на истинность предиката не влияет, поэтому его можно отбросить. P:(2*X=3*Y) and (X<=2*Y).

Проходим ветвь 2 от C к B и к предикату через конъюнкцию присоединяем условие B (здесь оно истинно). Получаем P: (2*X=3*Y) and (X<=2*Y) and (X<>Y).

В полученном предикате опять из первого члена конъюнкции следует третий, поэтому его можно отбросить. Получаем P: (2*X=3*Y) and (X<=2*Y).

Проходим ветвь 1 от B к A, при этом предикат пути не изменяется. Окончательно получаем P: (2*X=3*Y) and (X<=2*Y).

Для построения теста осталось подобрать значения X и Y так, чтобы предикат стал истинным. Это возможно, например, при

X= 3, Y=2 получим ответ 1;
X= 6, Y=4 получим ответ 2;
X= 9, Y=6 получим ответ 3;
X=12, Y=8 получим ответ 4 и т.д.

Упражнения:

1. Постройте предикат пути 1 - 2 - 5 - 2 - 5 - 2 - 4 - 3 и получите тест по построенному предикату.

2. Выполните трассировку и установите, какую задачу решает следующий фрагмент программы:

```
while x<>y do
begin     while x>y do x:=x-y;
          while y>x do y:=y-x;
end;
```

3. Сравните программы, рассмотренные в тексте параграфа и в упражнении номер два. Установите достоинства и недостатки каждой программы.

10.3.3. Поисковые циклы

Пример 10.9. Последовательность элементов задана формулой общего члена $a_i=\sin(i+i/n)$, где i изменяется от 1 до n (n - натуральное). Написать программу для нахождения первого элемента последовательности, большего заданного числа Z.

Решение. Исходными данными для решения задачи являются элементы последовательности: $a_1=\sin(1+1/n)$, $a_2=\sin(2+2/n)$, ..., $a_n=\sin(n+n/n)$. В результате получаем либо номер элемента, большего заданного Z, либо ответ: «Такого элемента нет».

Решение этой задачи может закончиться по двум причинам: первая - перебрали все элементы последовательности и не нашли нужного элемента; вторая - в процессе перебора обнаружился элемент, больший Z. В этом случае перебор прекращается и формируется ответ. Когда будет найден ответ неизвестно, следовательно, это поисковый цикл.

Первую причину окончания можно определить, проверив условие: i > n, где i - текущий элемент последовательности, а n - количество элементов в ней. Вторая причина имеет два значения: «найдено» или «не найдено», поэтому ее можно изображать логическим значением, где true соответствует найдено, а false - не найдено. Таким образом, условие окончания поиска может быть записано в виде (i>n) or f, что соответствует фразе русского языка: «Просмотрены все элементы или найдено». Просмотр элементов последовательности в цикле будет выполняться в случае ложности приведенного условия. Найдем его отрицание, воспользовавшись законом де Моргана: отрицание дизъюнкции равно конъюнкции отрицаний.

not((i>n) or f) = not(i>n) and not f = (i<=n) and not f.

Программа на алгоритмическом языке Паскаль:

```
program pr4;
{ поиск первого элемента последовательности, заданной формулой общего
члена a(i)=sin(i+i/n), где i=1,2,...,n, большего заданного Z }
var     n:integer; { количество элементов в последовательности }
        z:real; { заданное число }
        i:integer; { номер очередного элемента последовательности }
        f:boolean; { true, если найден искомый элемент }
begin
   write('Введите n и z ');
   readln(n,z);
   i:=1; f:=false;
   while (i<=n) and not f do { пока есть элементы и не найдено }
        if sin(i+i/n)>z
        then f:=true { искомый элемент найден }
        else i:=i+1; { переходим к следующему элементу }
   if f
   then writeln('Первый элемент больший ',z,' имеет номер ',i)
   else writeln('Среди элементов последовательности нет ',
                'больших ',z);
end.
```

Для тестирования подготовим три теста.

Тест 1: n=10, z=0.9. В этом случае получим ответ: «Первый элемент, больший 0.9 имеет номер 7».

Тест 2: n=10, z=2. В этом случае получим ответ: «Среди элементов последовательности нет больших 0.9».

Тест 3: n=-1, z=2. В этом случае получим ответ: «Среди элементов последовательности нет больших 2».

Упражнения:

1. Исправьте программу pr4 так, чтобы перебор элементов осуществлялся от n-го элемента к первому.

2. Исправьте программу pr4 так, чтобы осуществлялся поиск элемента, совпадающего с z, среди элементов этой же последовательности с номерами от p до q.

3. Перепишите программу pr4, используя для организации цикла оператор re-peat - until.

10.4. Модификации построенных алгоритмов

Мы написали несколько циклических программ. Чтобы сконструировать новые, можно применить приём модификации уже существующих алгоритмов. Будем использовать для модификации написанные программы, изменять операции, константы, переменные и наблюдать, что происходит с программой.

10.4.1. Модификация 1

Посмотрим, как изменится программа pr2 из подраздела 10.3.1, если в операторе s:=s+i изменить знак плюс на знак умножить. Чтобы понять, что делает приведенный ниже фрагмент программы, выполним трассировку.

```
{ фрагмент 1 }
readln(N);
s:=0; <-- начальное значение нужно изменить на 1.
i:=1;
while i<=N do
begin     s:=s*i; <-- Знак + изменили на *.
          i:=i+1;
end;
```

В приведенном фрагменте, поскольку s=0, результат получится равным нулю, но если изменить начальное значение s на 1, то в результате получится произведение натуральных чисел от 1 до N.

В начале $s=1, i=1$.
После первого шага $s=1\cdot1, i=2$.
После второго шага $s=1\cdot1\cdot2, i=3$.
После третьего шага $s=1\cdot1\cdot2\cdot3, i=4$.
После четвертого шага $s=1\cdot1\cdot2\cdot3\cdot4, i=5$.

Нетрудно догадаться, что после N шагов получим: $s=1\cdot1\cdot2\cdot3\cdot\ldots\cdot N, i=N+1$.

Произведение натуральных чисел от 1 до N называется факториалом числа и обозначается $N! = 1\cdot2\cdot3\cdot\ldots\cdot N$. По определению $0!=1$ и $1!=1$.

10.4.2. Модификация 2

Посмотрим, как изменится алгоритм pr2, если в операторе i:=i+1 изменить 1 на 2. Чтобы понять, что делает приведенный ниже фрагмент программы, выполним трассировку.

```
{ фрагмент 2 }
readln(N);
s:=0;
i:=1;
while i<=N do
begin  s:=s+i;
       i:=i+2; <-- 1 изменили на 2.
end;
```

В начале $s=0, i=1$.
После первого шага $s=0+1, i=3$.
После второго шага $s=0+1+3, i=5$.

После третьего шага s=0+1+3+5, i=7.
После четвертого шага s=0+1+3+5+7, i=9.

Отсюда следует, что после окончания цикла получим сумму нечетных натуральных чисел от 1 до N.

Для решения этой же задачи можно воспользоваться формулой нечетного числа, которая позволяет вычислить нечетное число по его номеру: нечетное (i) = 2·i-1. Получаем следующий фрагмент:

```
{ фрагмент 2а }
readln(N);
{ Здесь N - количество слагаемых }
s:=0;
i:=1;
while i<=N do
begin     s:=s+2*i-1;
          i:=i+1;
end;
```

Упражнение. Напишите программу для вычисления суммы s = 1/3 + 1/5 + ... + 1/(2·n+1).

Если необходимо вычислить сумму четных натуральных от 1 до N, то начать счет вычисления нужно с i=2. Получим следующий фрагмент:

```
 { фрагмент 3 }
 readln(N);
 s:=0;
 i:=2;
 while i<=N do
 begin     s:=s+i;
           i:=i+2;
 end;
```

В начале s=0, i=2.
После первого шага s=0+2, i=4.
После второго шага s=0+2+4, i=6.
После третьего шага s=0+2+4+6, i=8.
После четвертого шага s=0+2+4+6+8, i=10.

После окончания цикла получим сумму четных натуральных чисел от 1 до N.

Для решения этой же задачи можно воспользоваться формулой четного числа, которая позволяет вычислить четное число по его номеру: четное (i) = 2·i. Получаем следующий фрагмент:

```
{ фрагмент 3а }
readln(N);
{ Здесь N - количество слагаемых }
s:=0;
i:=1;
while i<=N do
begin     s:=s+2*i;
          i:=i+1;
end;
```

Упражнение. Напишите программу для вычисления суммы s = 1/2 + 1/4 + ... + +1/(2·n).

10.4.3. Модификация 3

Посмотрим, как изменится фрагмент 2, если в операторе i:=i+2 изменить знак + на знак *. Чтобы понять, что делает приведенный ниже фрагмент программы, выполним трассировку.

```
{ фрагмент 4 }
readln(N);
s:=0;
i:=1;
while i<=N do
begin   s:=s+i;
        i:=i*2; <-- + изменили на *.
end;
```

В начале s=0, i=1.
После первого шага s=0+1, i=2.
После второго шага s=0+1+2, i=4.
После третьего шага s=0+1+2+4, i=8.
После четвертого шага s=0+1+2+4+8, i=16.

Продолжая трассировку заключаем, что после окончания цикла получим сумму степеней числа два, меньших N.

10.4.4. Модификация 4

Посмотрим, как изменится фрагмент 2, если в операторе s:=s+i изменить i на некоторое заданное число x. Чтобы понять, что делает приведенный ниже фрагмент программы, выполним трассировку.

```
{ фрагмент 5 }
readln(N,x); <-- вводим число x. N - граница суммирования.
s:=0;
i:=1;
while i<=N do
begin   s:=s+x; <-- i изменили на x.
        i:=i+1;
end;
```

В начале s=0, i=1.
После первого шага s=0+x, i=2.
После второго шага s=0+x+x, i=3.
После третьего шага s=0+x+x+x, i=4.
После четвертого шага s=0+x+x+x+x, i=5.

Замечаем, что после окончания цикла получим сумму числа x, повторенного в качестве слагаемого N раз, т.е. x·N.

10.4.5. Модификация 5

Если во фрагменте 5 в операторе s:=s+x изменить знак + на знак * и начальное значение s изменим на 1, то в результате найдем N-ю степень числа x.

10.4.6. Модификация 6

Несложно изменить программу pr2 так, чтобы можно было найти сумму натуральных чисел на отрезке от a до b. Достаточно заметить, что программа pr2 решает эту задачу для случая a=1 и b=N. Заменив в программе pr2 1 на a и N на b, получим программу pr3, которая приводится ниже. Цикл в этой программе можно рассматривать как итерационный, т.к. сложение нужно выполнять пока не достигнем последнего слагаемого.

```
program pr3;
{ вычисление суммы натуральных чисел от a до b }
var         a,b:integer; { интервал суммирования - исходные данные }
            s:integer; { сумма - результат }
            i:integer; { слагаемое }
begin
   write('Введите интервал суммирования a<b ');
   readln(a,b); { Ввод исходных данных }
   s:=0; { Начинаем суммирование с нуля }
   i:=a; { Начальное значение слагаемого }
   while i<=b do { Нужно выполнять до тех пор, пока не достигнем последнего
                   слагаемого }
   begin  s:=s+i; { Добавили слагаемое к сумме }
          i:=i+1; { Увеличили слагаемое на единицу }
   end; { Закончилось тело цикла }
   write('Сумма чисел от ',a,' до ',b,' равна ',s) { Вывод результата }
end.
```

10.5. Задачи на построение циклов

Пример 10.10. Написать программу для нахождения суммы нечетных натуральных чисел на отрезке от a до b.

Например, для a=3 и b=9 или a=3 и b=10, или a=2 и b=9, или a=2 и b=10 в качестве ответа должно получиться одно и то же число: 3+5+7+9.

Решение 1. Эту задачу можно понять так, что заданы все числа отрезка от a до b. Из них необходимо в сумму включить только нечетные числа. Для решения задачи организуем перебор всех чисел отрезка от a до b. Для каждого выбранного числа проверяем, делится ли оно на 2, если не делится, то прибавляем его к сумме. В соответствии с этим получаем программу на Паскале:

```
program SUMMA1;
{ Сумма нечетных чисел отрезка от a до b }
var         a,b : integer; { границы отрезка }
            s : integer; { сумма }
            i : integer; { очередное число отрезка }
begin
   write('Введите границы отрезка [a,b] ');
   readln(a,b);
   s:=0; { начальное значение суммы }
```

```
    i:=a; { начальное число отрезка }
    while i<=b do { проверка: не закончен ли перебор }
    begin  if i mod 2 =1 { если i - нечетное, }
           then s:=s+i; { то добавить его к сумме }
           i:=i+1 { перейти к следующему числу отрезка }
    end;
    write('Сумма=',s)
end.
```

Тестирование. В построенной программе имеется три пути, ведущих от начала программы к ее концу. Следовательно, необходимо построить три теста так, чтобы по каждому пути пройти хотя бы по одному разу.

Первый путь: без захода в цикл. Тест: a=4, b=2. Ожидаемый ответ: s=0.

Второй путь: с заходом в цикл, но без выполнения условного оператора. Тест: a=4, b=4. Ожидаемый ответ: s=0.

Третий путь: с заходом в цикл и с выполнением условного оператора.
Тест: a=4, b=12. Ожидаемый ответ: s=32.

Решение 2. Эту задачу можно понять и так, что заданы только нечетные числа отрезка от a до b. Их необходимо просуммировать. Для решения задачи нужно найти первое нечетное число отрезка, а затем перебирать только нечетные числа, включая их в сумму. В соответствии с этим получаем программу на Паскале:

```
program SUMMA2;
{ Сумма нечетных чисел отрезка от a до b }
var     a,b : integer; { границы отрезка }
        s : integer; { сумма }
        i : integer; { очередное число отрезка }
begin
    write('Введите границы отрезка [a,b] ');
    readln(a,b);
    s:=0; { начальное значение суммы }
    if a mod 2 = 1 { если a - нечетное, }
    then i:=a { то начинаем перебор с a, }
    else i:=a+1; { иначе начинаем перебор со следующего числа }
    while i<=b do { проверка: не закончен ли перебор }
    begin
        s:=s+i; { добавить очередное число к сумме }
        i:=i+2 { перейти к следующему нечетному числу отрезка }
    end;
    write('Сумма=',s)
end.
```

Тестирование. В построенной программе имеется, по крайней мере, четыре пути, ведущих от начала программы к ее концу. Следовательно, необходимо построить не менее четырех тестов так, чтобы по каждому пути пройти хотя бы по одному разу.

Первый путь: a - нечетное, без захода в цикл. Тест: a=5, b=2. Ожидаемый ответ: s=0, т.к. отрезок задан неверно.

Второй путь: a - нечетное, с заходом в цикл. Тест: a=5, b=7. Ожидаемый ответ: s=12.

Третий путь: a - четное, без захода в цикл. Тест: a=4, b=1. Ожидаемый ответ: s=0, т.к. отрезок задан неверно.

Четвертый путь: a - четное, с заходом в цикл. Тест: a=4, b=7. Ожидаемый ответ: s=12.

Упражнения:

1. Решите эту задачу примера 10.10, перебирая числа в обратном порядке от b к a.

2. Если в программе summa2 заменить 2 на c и модифицировать оператор присваивания, стоящий после фразы else, то получится следующий фрагмент:

```
{ фрагмент 6 }
readln(a,b,c);
s:=0;
if a mod c = 0
then i:=a
else i:=a+c-a mod c;
while i<=b do
begin
    s:=s+i;
    i:=i+c
end;
write('Сумма=',s)
```

Какую задачу решает этот фрагмент, если c - натуральное число?

3. Аналогично задаче примера 10.10 решается задача нахождения суммы и произведения четных натуральных чисел на отрезке от [a;b]. Постройте программы самостоятельно.

4. Постройте также программы для нахождения произведения натуральных чисел на отрезке [a;b], произведения четных и нечетных натуральных чисел на отрезке [a;b].

Пример 10.11. Написать программу для вычисления суммы вида -1+2-3+4-..., в которой n слагаемых.

Решение 1. Установим, что повторяется в теле цикла. Здесь для нахождения суммы можно прибавлять слагаемые вида: s := s + z*i, где s - сумма, z - знак слагаемого, i - слагаемое. Знак числа изменяется на противоположный при каждом выполнении тела цикла. Тело цикла здесь выполняется n раз, поэтому цикл - арифметический. Соответствующий фрагмент приведен ниже:

```
{ фрагмент 7 }
write('Введите количество слагаемых ');
readln(n);
s:=0; { начальное значение суммы }
z:=-1; { знак первого слагаемого минус }
i:=1; { первое слагаемое }
while i<=n do { пока есть слагаемые }
begin      s:=s+z*i; { прибавляем слагаемое со знаком к сумме }
           z:=-z; { меняем знак }
```

```
            i:=i+1 { переходим к следующему слагаемому }
end;
```

Упражнение. Проведите трассировку фрагмента 7, затем ответьте на вопрос: «Какую задачу решает фрагмент 8, приведенный ниже?»

```
{ фрагмент 8 }
readln(a,b);
s:=0;
z:=-1;
i:=a;
while i<=b do
begin       s:=s+z*i;
            z:=-z;
            i:=i+1
end;
```

Решение 2. Можно заметить, что знак минус стоит перед нечетным слагаемым, а знак плюс перед - четным, поэтому в теле цикла повторяются следующие действия: если очередное слагаемое четное, то оно прибавляется к сумме, а если нечетное, то вычитается. Эти действия оформляются в виде условного оператора. Цикл по-прежнему арифметический. Тогда получаем фрагмент 9.

```
{ фрагмент 9 }
readln(n);
s:=0;
i:=1;
while i<=n do
begin
    if i mod 2=0 { если очередное слагаемое четное, }
    then s:=s+i { то оно прибавляется к сумме, }
    else s:=s-i; { иначе оно вычитается из суммы }
    i:=i+1
end;
```

Упражнение. Измените фрагмент 9 так, чтобы он решал ту же задачу, что и фрагмент 8.

Решение 3. Можно вначале найти сумму слагаемых, входящих в требуемую сумму со знаком минус, затем - сумму слагаемых, входящих в требуемую сумму со знаком плюс. Эти задачи уже решались нами (см. фрагменты 2 и 3 в подразделе 10.4.2). Затем из второй суммы вычтем первую и получим ответ для поставленной задачи. Решение приведено во фрагменте 10.

```
{ фрагмент 10 }
readln(N);
sn:=0; { сумма нечетных }
i:=1;
while i<=N do
begin       sn:=sn+i;
            i:=i+2;
end;
sc:=0; { сумма четных }
```

```
i:=2;
while i<=N do
begin     sc:=sc+i;
          i:=i+2;
end;
s:=sc-sn; { ответ задачи }.
```

Упражнение. Измените фрагмент 10 так, чтобы он решал задачу фрагмента 8.

Пример 10.12. Усложним пример 10.11. Пусть нужно найти сумму вида s = 1+2-3+4+5-6+... . В сумму входит n слагаемых.

Решение. Попытаемся применить предыдущее решение 3. Для этого нужно вначале сложить все числа, а затем сложить числа, кратные трем. Вычтя из первой суммы утроенную вторую, получим ответ задачи. В самом деле, например, для n=6 получим 1+2+3+4+5+6= 1+2+4+5+3+6-2·(3+6).

Несложно применить и второе решение предыдущей задачи для решения данной. Для этого достаточно из суммы вычитать числа, кратные трем, а остальные прибавлять. Получим фрагмент 11.

```
{ фрагмент 11 }
s:=0;
for i:=1 to n do
   if i mod 3=0
   then s:=s-i
   else s:=s+i.
```

Чуть сложнее применить первое решение предыдущей задачи для решения данной. Здесь нельзя изменять знак в цикле при каждом его исполнении. Возможные решения представлены в приведенных ниже фрагментах 12 и 13.

```
{ фрагмент 12 }
s:=0; { начальное значение суммы }
z:=1; { начальное значение знака + }
for i:=1 to n do { цикл выполняется n раз }
begin     if i mod 3=0 { если i кратно трем, }
          then z:=-1 { то знак слагаемого - }
          else z:=1; { иначе знак слагаемого + }
          s:=s+z*i { прибавляем к сумме очередное слагаемое с учетом знака}
end;

{ фрагмент 13 }
s:=0; { начальное значение суммы }
z:=1; { начальное значение знака + }
zk:=1; { начальное значение счетчика группы слагаемых }
for i:=1 to n do { цикл выполняется n раз }
begin     s:=s+z*i; { прибавляем к сумме очередное слагаемое с учетом знака }
          zk:=zk+1; { увеличиваем значение счетчика группы слагаемых }
          if zk=3 { если счетчик группы слагаемых равен трем, }
   then begin     zk:=0; { то сбрасываем счетчик в ноль }
                  z:=-1 { знак очередного слагаемого - }
          end
```

```
    else z:=1 { иначе знак очередного слагаемого + }
end;
```

10.6. Способы уяснения действий, составляющих тело цикла

Пример 10.13. Написать программу для вычисления выражения y = k(k+1)(k+2)...2k.

Решение. В данном примере исходным данным является k. Выражение представляет собой произведение. Перепишем его так, чтобы все сомножители имели одинаковую форму: (k+0)(k+1)(k+2)...(k+k). Исходя из этой формы, можно установить, что отыскивается произведение сомножителей. Чтобы установить общий вид сомножителя заметим, что каждый сомножитель содержит переменную k, к которой прибавляется некоторое натуральное число. Это число меняется от сомножителя к сомножителю, поэтому обозначим его переменной i. Отсюда общий вид сомножителя: (k+i), где i - изменяется от 0 до k. В теле цикла повторяется оператор вида y:=y*(k+i). Известно, что тело цикла должно выполниться k раз, следовательно, нужно построить арифметический цикл.

```
{ фрагмент 14 }
y:=1; {начальное значение произведения}
for i:=0 to k do y:=y*(k+i).
```

В данном примере использовался прием нахождения формулы общего члена на основе анализа структуры сомножителей.

Упражнения:

1.Используйте рассмотренный выше прием для вычисления значений по формулам:

1) $y=(k+m)(k+2m)...(k+m^2)$;

2) y=(k+3)(2k+6)(3k+9)..., всего в этой формуле должно быть n сомножителей.

2. Решите пример 10.13 и упражнения к нему, заменив умножение на сложение.

3. Придумайте еще примеры, которые решались бы аналогично.

Пример 10.14. Вычислить значение выражения: y=cos(1+cos(2+...+cos(n-1+cos(n)...)).

Решение. Для начала запишем заданную формулу при различных значениях n.
Для n=3 получим y=cos(1+cos(2+cos(3))).
Для n=5 получим y=cos(1+cos(2+cos(3+cos(4+cos(5))))).

Прежде всего заметим, что в заданном выражении используются натуральные числа. Обозначим их переменной i, которая изменяется от 1 до n, или, что одно и то же, от n до 1. Чтобы вычислить это выражение без компьютера, «вручную», нужно начать вычисления с самых вложенных скобок:

1. $y_1=cos(5)$
2. $y_2=cos(4+cos(5))$ или с учетом первого шага $cos(4+y_1)$
3. $y_3=cos(3+y_2)$
4. $y_4=cos(2+y_3)$
5. $y_5=cos(1+y_4)$

Если учесть обозначение i и переписать первый шаг как $y_1=cos(5+y_0)$, где $y_0=0$, то в теле цикла будет повторяться оператор y=cos(i+y). Количество повторений

известно заранее и равно n, следовательно, цикл - арифметический. Здесь в качестве счетчика можно использовать переменную i.

```
{ фрагмент 15 }
y:=0;
for i:=n downto 1 do y:=cos(i+y).
```

Пример 10.15. Вычислить для произвольного натурального n $y = \sqrt{n + \sqrt{n-1+\ldots+\sqrt{2+\sqrt{1}}}}$.

Решение. Этот пример очень похож на предыдущий, однако здесь вместо функции cos(x) используется квадратный корень и натуральные числа под корнями уменьшаются. Поскольку структура задач одинакова, они обе должны иметь одинаковые решения. Отсюда получаем фрагмент программы.

```
{ фрагмент 16 }
y:=0;
for i:=1 to n do y:=sqrt(i+y).
```

Пример 10.16. Вычислить выражение для n умножений. Например, для n=5 y=(((((x+a)x+a)x+a)x+a)x+a)x+a.

Решение. Этот пример также похож на предыдущий, где используется функция умножения, а вместо отрезка натурального ряда - константа a. Однако для начинающих обнаружить это сходство довольно трудно, поэтому установим оператор, повторяющийся в теле цикла, рассмотрев любые два соседних значения y, вычисленные «вручную».

$y_1=x+a$.

$y_2=(x+a)x+a$ или $y_2=y_1x+a$.

По индукции делаем вывод о том, что в теле цикла для получения нового значения нужно старое умножить на x и прибавить a. Если заметить, что $y_0=1$, $y_1=y_0x+a$, то тело цикла будет выполняться n+1 раз.

```
{ фрагмент 17 }
y:=1;
for i:=1 to n+1 do y:=y*x+a.
```

Другое решение можно получить, если заметить, что $y_0=x$, $y_1=(y_0+a)x$. В этом случае тело цикла выполняется n раз и после завершения цикла нужно прибавить a.

```
{ фрагмент 18 }
y:=x;
for i:=1 to n do y:=(y+a)*x;
y:=y+a.
```

Пример 10.17. Вычислить $y=1+x^n/(2+x^{n-1}/(3+x^{n-2}/(.../(n+x)...)$.

Решение. Этот пример также похож на предыдущие. В нем используются функция деления и натуральные числа от 1 до n. Аналогично предыдущим примерам, здесь повторяется следующий оператор: y=i+p/y, где p - степень x. Степень можно получить последовательными умножениями на x.

```
{ фрагмент 19 }
y:=1;
p:=1; { степени x }
for i:=n downto 1 do begin p:=p*x; y:=i+p/y end.
```

Упражнения:

1.Вычислите y=sin(n+sin(n-1+sin(...+sin(2+sin(1)...).

2. Вычислите y=(...(x-1)x-2)x-3)...)x-n.

3. Установите сходство и различия упражнений 1 и 2 и примеров 10.13 - 10.17. Установите, по какому принципу были придуманы эти упражнения. Придумайте на основе примеров 10.13 - 10.17 подобные упражнения и выполните их.

4. Вычислите выражение для n умножений. Например, для n=5 $y=(((((x+a)^2+a)^2+a)^2+a)^2+a)^2+a$.

Пример 10.18. Вычислить y=sin(x)+sin(sin(x))+...+sin(...sin(x)...), в последнем слагаемом функция вложена n раз.

Решение. Искомый результат является суммой, каждое слагаемое которой имеет вид sin(sin(sin(...sin(x)...), где sin повторен n раз. Прямолинейным решением этой задачи будет решение, в котором в тело цикла суммирования будет вложен цикл вычисления очередного слагаемого. Задача вычисления очередного слагаемого похожа на задачу примера 10.14, где функция cos заменена функцией sin и отсутствуют натуральные числа, поэтому решение будет таким:

```
{ фрагмент 20 }
y:=0; { начальное значение суммы }
for i:=1 to n do { в сумме n слагаемых }
begin     s:=x; { начальное значение для очередного слагаемого }
          for j:=1 to i do s:=sin(s);{ функция вкладывается i раз }
          y:=y+s
end.
```

Цикл, находящийся в теле другого цикла, называется *вложенным*. Использование вложенных циклов упрощает программирование, но увеличивает время решения задачи. Наличие вложенного цикла - недостаток фрагмента 20. От него можно избавиться, если заметить, что при вычислении очередного слагаемого можно использовать его предыдущее значение. Получаем фрагмент 21.

```
{ фрагмент 21 }
y:=0; { начальное значение суммы }
s:=x; { начальное значение слагаемого }
for i:=1 to n do
begin     s:=sin(s); { вычисление следующего слагаемого на основе
предыдущего }
          y:=y+s { вычисление суммы }
end.
```

Пример 10.19. Вычислить y = -sin(x) + cos(sin(x)) + sin(cos(sin(x))) -cos(sin(cos(sin(x))))+..., в сумме используется n слагаемых.

Решение. Оно основывается на решении примеров 10.12 и 10.18. Прежде всего заметим, что слагаемое состоит из знака и суперпозиции функций. Если с каждым слагаемым связать его порядковый номер, начав с 1, то можно заметить, что слагаемые с четным номером начинаются с косинуса, а слагаемые с нечетным номером - с синуса. У слагаемого стоит знак минус, если его порядковый номер при делении на 3 в остатке дает 1. В случае других остатков используется знак плюс. Заранее известно, что цикл выполняется n раз. Исходя из этого, можно построить фрагмент программы.

```
{ фрагмент 22 }
```

```
y:=0; { начальное значение суммы }
s:=x; { начальное значение слагаемого }
z:=-1; { начальное значение знака }
for i:=1 to n do
begin   if i mod 3=1 then z:=-1 else z:=1; { формируем знак }
        if i mod 2=1
        then s:=sin(s)
        else s:=cos(s); { формируем слагаемое }
        y:=y+z*s
end.
```

Упражнения:

1. Что получится, если во фрагменте 21 поменять местами операторы в теле цикла? Как исправить фрагмент, чтобы преобразованная программа работала верно? Какой вариант программы, на ваш взгляд, лучше и почему?

2. Вычислите $tg|2+tg|2^2+tg|2^3+tg|2^4+...+tg|2^n||...|$. Здесь n вложений модуля.

3. Вычислите $y=\sin(1+\cos(2-\sin(3+\cos(4+\sin(5-\cos(6+...)...))$. Здесь всего n вложений функций.

4. Вычислите $y = \cos(\sqrt{3+\sin(\sqrt{6+\cos(\sqrt{9+\sin(\sqrt{12+...)...)}}$. Здесь вложены n корней.

5. Вычислите $y = \sin(1) - \cos(\sin(2^2)) + \sin(\cos(\sin(3^3))) + \cos(\sin(\cos(\sin(4^4)))) - \sin(\cos(\sin(\cos(\sin(5^5))))) + ...$. Здесь всего n слагаемых.

6. Вычислите $y = \sqrt{\frac{1}{|\cos 2|}+\sqrt{\frac{|\sin 3|}{4}+\sqrt{\frac{5}{|\cos 6|}+\sqrt{\frac{|\sin 7|}{8}+...}}}}$ Здесь вложены n корней.

7. Вычислите $y=\sin(3/(\cos(3^2/(\sin(3^3/(\cos(3^4/(\sin(3^5/...)...))$. Здесь используется нечетное n пар скобок. В самых внутренних скобках нет деления.

8. Вычислите $y=1^n+2^n+3^n+4^n+...+n^n$.

9. Вычислите $y=n^1+n^2+n^3+n^4+...+n^n$.

10. Вычислите $y=1^1+2^2+3^3+4^4+...+n^n$.

Пример 10.20. Дано натуральное число n. Найти сумму цифр этого числа.

Решение. Воспользуемся информацией главы 8 «Линейные алгоритмы» о том, что для определения последней цифры числа нужно найти остаток от деления этого числа на 10, а для отбрасывания этой цифры - найти частное от деления целых чисел (операция div). В этом примере заранее неизвестно количество выполнений цикла, но существует одна причина его окончания - цифры числа n закончились, поэтому этот цикл итерационный. Переменной этого цикла является переменная, где хранится значение числа n, которое уменьшается от шага к шагу на одну цифру, а когда все цифры закончатся, закончится и цикл.

```
{ фрагмент 23 }
s:=0; { сумма цифр числа n }
while n>0 do { пока есть цифры в числе n }
begin   s:=s+n mod 10; { прибавить последнюю цифру к сумме }
        n:=n div 10 { отбросить последнюю цифру }
end.
```

Пример 10.21. Дано натуральное число n. Выяснить, входит ли цифра 2 в запись этого числа.

Решение. В теле цикла повторяются те же операции (найти и проанализировать последнюю цифру, отбросить ее), что и в предыдущем примере. Однако причин окончания цикла может быть две: цифра 2 входит в запись числа n и цифра 2 не входит в запись этого числа. Следовательно, этот цикл должен быть построен как поисковый с проверкой каждой из двух причин его окончания.

```
{ фрагмент 24 }
f:=false;
while (n>0) and not f do
    if n mod 10 =2 then f:=true else n:=n div 10;
if f then write('цифра 2 входит в число ')
else write('цифра 2 НЕ входит в запись числа').
```

Пример 10.22. Дано натуральное число n. Поменять порядок цифр числа n на обратный.

Решение. Например, для n=1829 должно получиться m=9281. Здесь используются те же операции, что и в примерах 10.20 и 10.21. Но последовательно получаемые цифры числа не отбрасываются, а результат m домножается на 10 и цифра прибавляется к результату. Цикл выполняется до тех пор, пока есть цифры в исходном числе.

```
{ фрагмент 25 }
m:=0; { в начале результата нет }
while n>0 do
begin     m:=m*10+n mod 10;
          n:=n div 10
end.
```

Пример 10.23. Вычислить

$$y = n!! = \begin{cases} 1 \cdot 3 \cdot 5 \cdot \ldots \cdot n, & \text{æ}^{\circ} \text{Ł } n - \text{Œ} \\ 2 \cdot 4 \cdot 6 \cdot \ldots \cdot n, & \text{æ}^{\circ} \text{Ł } n - \text{Œ} \end{cases}.$$

Решение. Заданная формула состоит из двух строк, каждая из которых представляет собой произведение. Следовательно, для решения этой задачи нужно использовать алгоритм нахождения произведения. Можно заметить, что начальное значение произведения начинается либо с 1, либо с 2 в зависимости от четности n. В этом случае получаем решение:

```
{ фрагмент 26 }
    if n mod 2=0 { начальное значение числа устанавливается }
    then i:=2 { на основе четности n }
    else i:=1;
    y:=1; { начальное значение двойного факториала }
    while i<=n do
    begin     y:=y*i;
              i:=i+2
    end.
```

Другое решение этого примера можно получить, если заметить, что произведение в любом случае заканчивается n. Если начать умножать с него и уменьшать затем число на два, получаем:

```
{ фрагмент 27 }
i:=n;
y:=1;
while i>1 do { от умножения на 1 произведение не изменяется }
begin     y:=y*i;
          i:=i-2
end.
```

Упражнения:

1. Дано натуральное число n. Найдите первую слева цифру в записи этого числа.

2. Дано натуральное число n. Найдите сумму цифр этого числа. Первая слева цифра к сумме прибавляется, следующая вычитается, третья слева цифра снова прибавляется и т.д.

3. Дано натуральное число n. Найдите сумму последних m цифр числа.

4. Дано натуральное число n. Поменяйте местами первую и последнюю цифры числа.

5. Дано натуральное число n. Проверьте, стоят ли в числе n две цифры 2 рядом.

6. Дано натуральное число n. Проверьте, стоят ли в числе n две четные цифры рядом.

7. Дано натуральное число n. Проверьте, стоят ли в числе n четная и нечетная цифры рядом.

8. Дано натуральное число n. Припишите цифру 2 в начало и в конец числа.

9. Дано натуральное число n. Проверьте, является ли это число палиндромом, т. е. читается ли оно одинаково слева направо и справа налево.

10. Вычислите

$$y = n!!! = \begin{cases} 1\cdot 4\cdot 7\cdot \ldots \cdot n, & \text{æ°Ł n Ł ° ŁŁ 3 , æ Œ1,} \\ 2\cdot 5\cdot 8\cdot \ldots \cdot n, & \text{æ°Ł n Ł ° ŁŁ 3 , æ Œ2,} \\ 3\cdot 6\cdot 9\cdot \ldots \cdot n, & \text{æ°Ł n °Ł æ 3.} \end{cases}$$

11. Обобщите упражнение 10 на случай нахождения m-факториала числа n. Для этого в приведенном выше определении замените 3 на m и добавьте недостающие строки.

10.7. Рекуррентные формулы

10.7.1. Основные понятия

Рекуррентной называется формула, связывающая значения p+1 соседних членов u_k, u_{k-1}, ..., u_{k-p} (k>=p+1) некоторой последовательности $\{u_n\}$ (n=1, 2, ...): $u_k = F(k,u_{k-1},...,u_{k-p})$. Рекуррентная формула позволяет шаг за шагом определить любой член последовательности, если известны p первых ее членов u_1, u_2,..., u_p, где p называют порядком формулы. Рассмотрим примеры.

Пример 10.24. $u_1=u_2=u_3=1$, для k>3 каждый следующий член последовательности определяется формулой $u_k=u_{k-1}+2\cdot u_{k-2}+3\cdot u_{k-3}$. Эту формулу называют линейной однородной формулой третьего порядка.

Пример 10.25. $u_1=u_2=1$, для k>2 каждый следующий член последовательности определяется формулой $u_k=u_{k-1}+u_{k-2}$. Эту формулу называют линейной однородной

формулой второго порядка. Она порождает числа Фибоначчи, каждое из которых равно сумме двух предыдущих.

Пример 10.26. $u_1=1$, $u_2=2$, для k>2 каждый следующий член последовательности определяется формулой $u_k=u_{k-2}$. Эту формулу называют линейной однородной формулой второго порядка. Обратите внимание на то, что элемент u_{k-1} здесь пропущен, но этот факт не снижает порядка формул.

Пример 10.27. $u_1=1$, для k>1 каждый следующий член последовательности определяется формулой $u_k=-u_{k-1}$. Эту формулу называют линейной однородной формулой первого порядка. Сравните эту формулу с формулой из примера 10.26).

Пример 10.28. $u_1=0$, $u_2=1$, для k>2 каждый следующий член последовательности определяется формулой $u_k = u_{k-1}^2 + \cos u_{k-2}$. Эту формулу называют нелинейной рекуррентной формулой второго порядка.

Пример 10.29. $u_1=0.5$, для k>1 каждый следующий член последовательности определяется формулой $u_k=u_{k-1}+3$. Эту формулу называют линейной неоднородной формулой первого порядка.

Пример 10.30. $u_1=1$, для k>1 каждый следующий член последовательности определяется формулой $u_k=k\cdot u_{k-1}$. Эту формулу называют линейной рекуррентной формулой первого порядка с переменными коэффициентами.

Пример 10.31. Рассмотрим задачу нахождения n-го члена рекуррентной последовательности на примере чисел Фибоначчи. Они определяются формулой примера 10.25. Каждое число Фибоначчи равно сумме двух предыдущих. В частности: $u_3=u_2+u_1=1+1=2$;
$u_4=u_3+u_2=1+2=3$;
$u_5=u_4+u_3=2+3=5$;
$u_6=u_5+u_4=3+5=8$ и т.д.

Отсюда следует, что для получения очередного числа достаточно хранить два предыдущих. Таким образом, в программе постоянно используются три соседних числа Фибоначчи. Для их хранения достаточно ввести три переменные: A хранит u_k, B хранит u_{k-1}, C хранит u_{k-2}. Для вычисления следующего числа Фибоначчи необходимо провести сдвиг, т.е. переписать содержимое B в C, а содержимое A в B. Исходя из этого, получаем фрагмент 28.

```
{ фрагмент 28 }
c:=1; { значение первого числа известно }
b:=1; { значение второго числа тоже известно }
k:=3; { начинаем вычисление с третьего числа }
while k<=n do { цикл, пока не найдено n-е число }
begin   a:=b+c;{вычисляем следующее число как сумму двух предыдущих }
        c:=b; { сдвигаем b в c для нахождения следующего числа }
        b:=a; { сдвигаем a в b для нахождения следующего числа }
        k:=k+1 { увеличиваем счетчик найденных чисел }
end;
write(a); { печатаем найденное число }
```

Упражнения:

1. Выполните трассировку фрагмента 28 для n=2,3,4.

2. Дополните фрагмент 28 так, чтобы он правильно выдавал числа Фибоначчи для любого n>=1.

Пример 10.32. Пусть необходимо разработать фрагмент программы для вычисления n-го члена последовательности, заданной формулой примера 10.26.

Решение. Первые элементы этой последовательности будут иметь вид: $u_1=1$; $u_2=2$; $u_3=u_2=1$; $u_4=u_2=2$; $u_5=u_3=1$ и т.д. Хотя в формуле используется два члена, хранить нужно три соседних члена, поэтому используем следующие переменные: A хранит u_k, B хранит u_{k-1}, C хранит u_{k-2}. Программа представлена в фрагменте 29.

```
{ фрагмент 29 }
c:=1;
b:=2;
k:=3;
while k<=n do
begin     a:=c;
          c:=b;
          b:=a;
          k:=k+1
end;
write(a);
```

Упражнение. Решает ли фрагмент 29a ту же задачу, что и фрагмент 29? Почему?

```
{ фрагмент 29a }
if n mod 2 =0
then a:=2
else a:=1;
```

10.7.2. Получение рекуррентных формул для последовательностей

Довольно часто в практике программирования возникают задачи вычисления значений различных конечных сумм: полиномов, отрезков степенных рядов, коэффициенты которых заданы явными выражениями в зависимости от номера члена, и т.п.

Если формула общего члена последовательности содержит мультипликативные операции, то для того, чтобы получить формулу, по которой следующий член последовательности вычисляется на основе предыдущего, нужно найти отношение двух соседних членов последовательности. Рассмотрим пример.

Пример 10.33. Вычислить значение многочлена (отрезка степенного ряда для e^x).

$$y = \sum_{i=0}^{n} \frac{x^i}{i!} = 1 + x + \frac{x^2}{1\cdot 2} + \ldots + \frac{x^n}{1\cdot 2\cdot \ldots \cdot n},$$

здесь x и n - заданные числа.

Решение. Обозначим общий член ряда через u_i, так что

$$y = \sum_{i=0}^{n} u_i, \quad u_i = \frac{x^i}{i!}.$$

Найдем отношение

$$\frac{u_i}{u_{i-1}} = \frac{x^i \cdot (i-1)!}{i! \cdot x^{i-1}} = \frac{x}{i}, \quad \text{æ} \quad u_i = u_{i-1} \cdot \frac{x}{i}.$$

Таким образом, зная член u_{i-1}, можно вычислить член u_i, выполнив всего две операции. Получаем фрагмент 30 программы.

```
{ фрагмент 30 }
y:=0;                    {начальное значение суммы }
u:=1;                    {первое слагаемое определим, поставив i=0 в формулу }
i:=1;                    {вычисления начинаются с первого члена ряда}
while i<=n do    {цикл, пока не получено n членов ряда }
begin     y:=y+u;        {добавить очередной член к сумме }
          u:=u*x/i;      {вычислить следующий член на основе предыдущего }
          i:=i+1         {изменить номер члена ряда }
end;
```

Пример 10.34. Вычислить значение многочлена (отрезка степенного ряда для e^x) с точностью до ε.

$$y = \sum_{i=0} \frac{x^i}{i!} = 1 + x + \frac{x^2}{1 \cdot 2} + \ldots;$$ здесь x и ε - заданные числа.

Решение. Вычислить сумму с точностью до ε, значит, продолжать суммирование очередного слагаемого до тех пор, пока оно больше ε. Поскольку заранее неизвестно, сколько раз выполняется тело цикла, то в данном случае имеем итерационный цикл. Используя результаты предыдущего примера, получаем фрагмент 31.

```
{ фрагмент 31 }
read(x,eps);
y:=0;
u:=1;
i:=2;
while u>eps do
begin     y:=y+u;
          u:=u*x/i;
          i:=i+1
end;
write(y);
```

Пример 10.35. Вычислить сумму отрезка степенного ряда для sin x:

$$\sin x \approx \sum_{i=0}^{n} (-1)^i \frac{x^{2i+1}}{(2i+1)!},$$ a x Ł n - ß.

Решение. Обозначим

$$u_i = \frac{(-1)^i x^{2i+1}}{(2i+1)!}.$$

Найдем отношение

$$\frac{u_i}{u_{i-1}} = \frac{(-1)^i x^{2i+1}(2i-1)!}{(2i+1)!(-1)^{i-1} x^{2i-1}} = \frac{-x^2}{2i(2i+1)}.$$

Отсюда

$$u_i = u_{i-1} \cdot \frac{-x^2}{2i(2i+1)}.$$

Упростим вычисление знаменателя. Обозначим его через r_i, тогда $r_i = 2i(2i+1) = 4i^2+2i$. Выразим r_i через r_{i-1}. Рассмотрим разность $d_i = r_i - r_{i-1} = 4i^2+2i-4(i-1)^2-2(i-1) = 8i-2$, откуда $r_i=r_{i-1}+d_i$. Теперь выразим d_i через d_{i-1}: $d_i-d_{i-1} = 8i-2-8(i-1)+2 = 8$ или $d_i = d_{i-1}+8$.

Используя полученные формулы, построим фрагмент программы для решения поставленной задачи.

```
{ фрагмент 32 }
d:=-2;
r:=0;
u:=x;
s:=0;
v:=-sqr(x);
while abs(u)>eps do { слагаемое может быть отрицательным }
begin     s:=s+u;
          d:=d+8;
          r:=r+d;
          u:=u*v/r
end.
```

Упражнения:

1. Написать фрагмент программы для вычисления с точностью до eps>0: $y=\operatorname{arctg} x=x-x^3/3+x^5/5-...+(-1)^n x^{2n+1}/(2n+1)+...$ ($|x|<1$).

2. Для больших значений |x| формула примера 10.34 малопригодна для вычисления e^x, так как члены ряда сначала возрастают по абсолютной величине и лишь потом начинают убывать. Быстрее и точнее значение e^x при $x=n+m$, где n - целое, а $|m|<=1/2$, можно вычислить по формуле $e^x=e^n \cdot e^m$, где e^n вычисляется последовательными умножениями или делениями на e, а e^m - с помощью ряда примера 2. Напишите фрагмент программы для вычисления e^x с помощью описанного метода.

Пример 10.36. Баржа, двигаясь по реке, ежедневно проходит расстояние, определяемое формулой

$$y = \begin{cases} y_{i-1} + 0.5 \cdot y_{i-2}, & \text{если i -четное,} \\ 0.7 \cdot y_{i-5} - 0.2 \cdot y_{i-4}; & \text{если i - нечетное.} \end{cases}$$

Определить, сможет ли баржа за n дней пройти расстояние в z километров, если в каждый из первых пяти дней она проходила по одному километру.

Решение. Для того чтобы вычислить дневной путь баржи, начиная с шестого дня, необходимо знать расстояния, пройденные баржей за предыдущие пять дней. Введем обозначения: y_i - a, y_{i-1} - b, y_{i-2} - c, y_{i-3} - d, y_{i-4} - e, y_{i-5} - f. Эта часть задачи

решается стандартно для любой задачи на рекуррентные формулы. Кроме того, необходимо найти путь баржи за n дней (сумма расстояний) и сравнить его с z.

```
{ фрагмент 33 }
var     a,b,c,d,e,f:real; { расстояния, пройденные баржей за 6 дней }
        i,n:integer; {i - счетчик, n - количество дней }
        s:real; { сумма расстояний, пройденных за n дней }
        z:real; { расстояние для сравнения }
begin
   f:=1;
   e:=1;
   d:=1;
   c:=1;
   b:=1;
   s:=5;
   write('Введите количество дней ');readln(n);
   for i:=5 to n do
   begin  if i mod 2=0
          then a:=b+0.5*c
          else a:=0.7*f-0.2*e;
          f:=e;
          e:=d;
          d:=c;
          c:=b;
          b:=a;
          s:=s+a;
   end;
   if s>z
   then write('Пройденное расстояние ',s,' больше ',z)
   else   if s=z
          then write('Пройденное расстояние ',s,' равно ',z)
          else write('Пройденное расстояние ',s,' меньше ',z)
end.
```

Пример 10.37. Найти максимальный элемент и его номер в последовательности, заданной формулой общего члена a(i)=sin(i+n/i), где i изменяется от 1 до n.

Решение. Первый элемент последовательности объявляем кандидатом на максимум. Последовательно сравниваем все остальные элементы последовательности с кандидатом на максимум. Если очередной элемент меньше или равен кандидату, то переходим к следующему элементу. Если очередной элемент больше кандидата, то заменяем им кандидата и продолжаем сравнения. Повторять сравнения не нужно, т.к. все предшествующие элементы были меньше или равны предыдущему кандидату, а он оказался меньше нового кандидата.

Программа на алгоритмическом языке Паскаль:

```
program primer;
{ Найти максимальный элемент последовательности, заданной формулой
общего члена, и его номер }
```

```
var       n :integer;{исходное данное - количество элементов в
последовательности}
          i :integer;{номер элемента последовательности}
          a :real; {элемент последовательности}
          k :real; {максимальный элемент}
          ki:integer;{номер максимального элемента}
begin
   write('введите число ');
   readln(n);
   k:=sin(1+n/1);ki:=1; {начальное формирование кандидата и его номера }
   i:=2;
   while i<=n do
   begin  a:=sin(i+n/i);
          if k<a  { Если кандидат меньше очередного элемента, }
          then begin k:=a;ki:=i end; { то запомнить его и его номер }
          i:=i+1;
   end;
   write('max=',k:15:2,' номер max=',ki);
end.
```

Для n=9 получаем ответ: max= 0.94 номер max=6.

Упражнения:

1. Ежедневная выручка лавочника равна половине выручки предыдущего дня, плюс четверть выручки второго предыдущего дня, плюс 0.15 выручки третьего предыдущего дня. Определите, через сколько лет разбогатеет лавочник, если по его понятиям богатство начинается с 1000 $, а выручка первых трех дней составила по 1 $ в день.

2. Прибыль купца за год, начиная с пятого, определяется формулой

$$p_i = \begin{cases} 3 \cdot p_{i-1} - 2 \cdot p_{i-3} + 20 & \text{если i - четное,} \\ 2 \cdot p_{i-2} - p_{i-4} + 15, & \text{если i - нечетное.} \end{cases}$$

Определите, на сколько процентов ежегодно после четырех лет богатеет купец, если первоначальный капитал его равнялся десяти тысячам рублей, а прибыль за первые четыре года составила 1, 2, 3, и 4 тысячи рублей соответственно.

3. Найдите минимальный член последовательности, заданный формулой общего члена $a_n=n^2-17n+21$. Найдите также номер этого члена.

4. Последовательность задана формулой общего члена $a_n=2^n+3^n$. Проверьте, выполняется ли равенство $(a_{n+1}+a_{n+2})/6-a_n=3^n$ для m первых членов последовательности.

5. Последовательность задана рекуррентно: $a_1=0$, $a_{2n}=a_{2n-1}+k$, $a_{2n+1}=k*a_{2n}$, где k - действительное число ($k<>0$; $k<>1$). Проверьте, что для m первых членов этой последовательности выполняется равенство $a_{2n}=(k_{n+1}-k)/(k-1)$.

6. Последовательность (a_n) задана формулой $a_n=2n^2-11n+442$. Последовательность (b_n) задана формулами: $b_n=2b_{n-1}-b_{n-2}+4$, $b_1=433$, $b_2=428$. Определите, совпадают ли первые m членов этих последовательностей. Определите какой последовательности принадлежит число Z и какой номер этого числа в последовательности.

10.8. Вложенные циклы

Циклы, размещенные в теле другого цикла, называются *вложенными*. Для правильной организации таких циклов нужно применить рассмотренную ранее последовательность действий к каждому циклу в отдельности. Очевидно, что параметры (переменные) этих циклов должны быть различными, в противном случае один цикл может нарушить работу другого. Программисты давно заметили, что для решения любой задачи достаточно только одного цикла. Однако, такая конструкция получается громоздкой и трудночитаемой. Поэтому при программировании часто используются вложенные циклы.

Пример 10.38. Последовательность (a_n) задается так: a_1 - некоторое натуральное число, a_{n+1} - сумма квадратов цифр числа a_n, n>=1. Найти m-й член последовательности.

Решение. Для решения задачи нужно m раз выполнить вычисление очередного члена последовательности. Для его вычисления необходим еще один цикл, в котором цифры последовательно отрываются от очередного члена последовательности и находится сумма их квадратов. В результате получаем фрагмент с вложенными циклами.

```
{ фрагмент 34 }
var         a, { член последовательности }
            s:integer; { сумма квадратов цифр предыдущего члена
последовательности}
            i, { счетчик }
            m:integer; { номер последнего члена последовательности }
begin
   write('Введите 1-й член последовательности и m ');
   readln(a,m);
   for i:=2 to m do
   begin  s:=0; { находим сумму квадратов цифр }
          while a>0 do
          begin  s:=s+sqr(a mod 10);
                 a:=a div 10
          end;
          a:=s { запоминаем новый член последовательности }
   end;
end.
```

Пример 10.39. Вычислить s=1·2+2·3·4+3·4·5·6+...+n·(n-1)·...·2·n.

Решение 1. Нужно вычислить сумму, которая состоит из n слагаемых. Каждое из слагаемых представляет собой произведение натуральных чисел. Если каждому слагаемому поставить в соответствие его порядковый номер, то можно заметить, что первый сомножитель любого слагаемого совпадает с его порядковым номером, последний - равен удвоенному порядковому номеру слагаемого. Решение можно получить, проделав два шага. На первом шаге (фрагмент 35) вычисляется сумма:

```
{ фрагмент 35 }
s:=0;
i:=1;
while i<=n do
```

```
begin { вычисление очередного слагаемого p }
    s:=s+p;
    i:=i+1
end.
```

Вычисление очередного слагаемого - это вычисление произведения $p=i\cdot(i+1)\cdot\ldots\cdot(2\cdot i)$ (фрагмент 36):

```
{ фрагмент 36 }
p:=1;
j:=i; { переменная цикла вычисления произведения }
while j<=2*i do
begin      p:=p*j;
           j:=j+1
end.
```

Для получения решения исходной задачи необходимо подставить фрагмент 36 в фрагмент 35 вместо комментария. Получим фрагмент 37.

```
{ фрагмент 37 }
s:=0;
i:=1;
while i<=n do
begin
    p:=1;
    j:=i; { переменная цикла вычисления произведения }
    while j<=2*i do
    begin
           p:=p*j;
           j:=j+1
    end;
    s:=s+p;
    i:=i+1
end.
```

Решение 2. Эту задачу можно решить и без использования вложенных циклов. Заметим, что при вычислении следующего слагаемого выполняется часть работы, которая уже была выполнена для предыдущего слагаемого. Выпишем i-е слагаемое: $u_i=i\cdot(i+1)\cdot\ldots\cdot(2\cdot i)$. (i-1)-е слагаемое равно: $u_{i-1} = (i-1)\cdot i\cdot(i+1)\cdot\ldots\cdot 2\cdot(i-1)$. Найдем их отношение:

$$\frac{u_i}{u_{i-1}} = \frac{i\cdot(i+1)\cdot(i+2)\cdot\ldots\cdot(2\cdot i-2)\cdot(2\cdot i-1)\cdot 2\cdot i}{(i-1)\cdot i\cdot(i+1)\cdot\ldots\cdot(2\cdot i-2)} = \frac{(2\cdot i-1)\cdot 2\cdot i}{i-1}.$$

Отсюда $u_i = u_{i-1}\cdot(2\cdot i-1)\cdot 2\cdot i/(i-1)$. Получаем фрагмент 38.

```
{ фрагмент 38 }
s:=0;
u:=2; { первое слагаемое }
i:=2; { номер очередного слагаемого }
while i<=n do
begin
    s:=s+u;
    u:=u*(2*i-1)*2*i/(i-1);
```

```
    i:=i+1
end.
```

Упражнения:

1. Установите начальные значения в следующей модификации фрагмента 38:

```
s:=  ;
u:=  ; { первое слагаемое }
i:=  ; { номер очередного слагаемого }
while i<=n do
begin
    u:=u*(2*i-1)*2*i/(i-1);
    s:=s+u;
    i:=i+1
end.
```

2. Вычислите:

а) s=1·2+3·4·5+6·7·8·9+..., всего n слагаемых;

б) s=(1+2) ·(3+4+5) ·(6+7+8+9) ·..., всего n сомножителей.

Сравните решение этих задач.

10.9. Стратегии ввода данных и вывода результатов

Для решения задач необходимо ввести исходные данные, т.е. взять их у человека и перенести в оперативную память компьютера. Для отдельных данных эту работу выполняет оператор ввода. Иногда требуется ввести в компьютер последовательность значений. Здесь возможны следующие ситуации. В первой ситуации требуется ввести заранее известное количество данных. Для этого организуют арифметический цикл со счетчиком, по значению которого заканчивается цикл ввода. Во второй ситуации заранее неизвестно количество вводимых элементов. Определить окончание ввода в этом случае можно одним из следующих способов. Можно установить последний элемент последовательности, после обработки которого ввод заканчивается, или ввести специальный признак, который будет сигнализировать об окончании исходных данных. Вторая ситуация реализуется при программировании итерационным циклом. Далее на примерах решения задач рассматриваются эти ситуации.

Пример 10.40. Возрастающая последовательность натуральных чисел вводится с клавиатуры до ввода числа 0. Вычислить и напечатать факториал каждого введенного числа.

Решение. Здесь признаком окончания ввода является число 0, заранее количество вводимых данных неизвестно, поэтому организуем итерационный цикл. Исходные данные для этой задачи могут быть такими: 2 4 7 9 13 0. Нужно получить: 2! 4! 7! 9! 13! Заметим, что для вычисления факториала следующего числа можно использовать факториал предыдущего числа (по условию числа возрастают!). Введем следующие обозначения: a - введенное число, b - число, с которого нужно продолжить вычисление факториала, i - счетчик выполненных умножений, p - факториал числа.

```
{ фрагмент 39 }
p:=1;
b:=1;
```

```
read(a);
while a<>0 do
begin
    for i:=b to a do p:=p*i;
    write('факториал ',a,' = ',p,' ');
    b:=a+1;
    read(a)
end.
```

Упражнения:

1. Зачем в программе использован оператор b:=a+1? Можно ли его было выполнить после оператора ввода?

2. Убывающая последовательность натуральных чисел вводится с клавиатуры до тех пор, пока не будет введен 0. Вычислите и напечатайте значение два, возведенное в степень, равную введенному числу. Вычисления должны быть эффективными, без повторов.

Пример 10.41. Последовательность из n положительных чисел вводится с клавиатуры. Найти сумму максимального и минимального элементов последовательности: x_1, x_2+1/x_1, x_3+1/x_2, ... , x_n+1/x_{n-1}, где x_i - введенное число.

Решение. Для ввода данных нужно организовать арифметический цикл по счетчику до ввода n чисел. Для работы на каждом шаге цикла нужно хранить два соседних числа. Начальным значением для максимума и минимума может быть значение x_1.

```
{ фрагмент 40 }
read(a); { вводим первое число последовательности }
max:=a;
min:=a;
for i:=2 to n do
begin
    read(b);
    c:=b+1/a; { вычисляем элемент заданной последовательности }
    if max<c then max:=c { находим max и min }
    else     if min>c then min:=c;
    a:=b { запоминаем предыдущий элемент }
end;
write(max+min).
```

Упражнение. Выясните, изменится ли работа фрагмента 40, если в условном операторе заменить «else» на «;». Какое решение лучше: с else или без else, почему?

Пример 10.42. Последовательность целых чисел вводится с клавиатуры. Первое введенное число не является членом последовательности, а задает количество членов последовательности. Определить, что встретится раньше в этой последовательности - положительное или отрицательное число.

Решение. В данной задаче количество вводимых данных определяется первым введенным элементом. После этого количество элементов для ввода будет известно, однако количество выполнения тела цикла по-прежнему неизвестно, так как все элементы могут оказаться равными нулю, тогда придется ввести все

элементы. Если при вводе встретится элемент, отличный от нуля, то результат будет известен и ввод можно завершить. Таким образом, имеется несколько причин окончания цикла, поэтому фрагмент должен предусматривать соответствующие варианты его окончания.

```
{ фрагмент 41 }
read(n);
i:=1;
f:=false;
while (i<=n) and not f do
begin
    read(a);
    if a=0 then i:=i+1 else f:=true
end;
if f
then        if a<0
            then write('отрицательный элемент встречается раньше')
            else write('положительный элемент встречается раньше')
else write('все элементы равны нулю').
```

Упражнение. Что нужно добавить к фрагменту 41, чтобы узнать номер первого положительного или отрицательного числа?

Пример 10.43. Последовательность чисел вводится с клавиатуры. Последним вводимым числом является число 100. Вычислить $s = x_1x_2x_3 + x_2x_3x_4 + x_3x_4x_5 + ... + x_{n-2}x_{n-1}x_n$, где $x_n=100$.

Решение. Количество элементов для ввода неизвестно, но окончанием ввода является обработка элемента, равного 100, поэтому цикл - итерационный. В цикле каждый раз обрабатываются три элемента, поэтому нужно принять соответствующие меры для их сохранения.

```
{ фрагмент 42 }
read(a,b); { читаем два первых элемента }
s:=0; { начальное значение суммы }
f:=(a<>100) and (b<>100); { a и b не равны 100? }
while f do { пока не обработан элемент равный 100 }
begin
    read(c); { читаем следующий элемент }
    s:=s+a*b*c; { вычисляем сумму }
    a:=b; { сдвигаем прочтенные
    b:=c; элементы }
    f:=c<>100 { элемент c не равен 100? }
end.
```

При выводе на печать следует помнить, что в стандартном Паскале строки выводятся (печатаются) последовательно. Напечатав строку, к ней нельзя вернуться. При выводе очень часто структуру алгоритма подсказывает структура печатаемых данных.

Пример 10.44. Напечатать квадрат со стороной из n звездочек (n>2).
Например, для n=4 должно получиться

```
****
```

****
****
****.

Решение. Выводимая информация состоит из n строк. Ее можно вывести с помощью арифметического цикла.

```
{ фрагмент 43 }
for i:=1 to n do
begin
   writeln; { в начало новой строки }
   { вывести i-ю строку }
end.
```

Вывод i-й строки состоит в печати n звездочек. Он может быть выполнен так:

```
{ фрагмент 44 }
{ вывести i-ю строку }
for j:=1 to n do write('*');
```

Если фрагмент 44 подставить вместо соответствующего комментария во фрагмент 43, то получим решение задачи:

```
{ фрагмент 45 }
for i:=1 to n do
begin
   writeln;  { в начало новой строки }
   for j:=1 to n do write('*') { вывести i-ю строку }
end.
```

Пример 10.45. Напечатать квадрат со стороной из n звездочек (n>2), причем звездочки должны располагаться по периметру квадрата.

Например, для n=4 должно получиться

```
* * * *
*     *
*     *
* * * *
```

Решение. Вначале разберемся со структурой (строением) выходной информации с учетом последовательной печати. С этой точки зрения квадрат состоит их трех частей. Первая и последняя строки одинаковы и состоят из n звездочек. n-2 средних строки также имеют одинаковую структуру и начинаются и кончаются звездочками, разделенными n-2 пробелами. Установленная структура определяет структуру алгоритма вывода квадрата:

1. Вывод строки из n звездочек.
2. Вывод n-2 строк.
3. Вывод строки из n звездочек.

Второй пункт детализируется так же с учетом структуры выводимых строк:

2. Цикл повторить n-2 раза:
    2.1. Вывод звездочки.
    2.2. Вывод n-2 пробелов.
    2.3. Вывод звездочки.

Расшифровывая второй пункт и заменяя фразы русского языка операторами Паскаля, получаем фрагмент программы.

```
{ фрагмент 46 }
{ 1 } writeln; for i:=1 to n do write('*');
{ 2 } for i:=1 to n-2 do
begin
    { 2.1 } writeln; write('*');
    { 2.2 } for j:=1 to n-2 do write(' ');
    { 2.3 } write('*')
end;
{ 3 } writeln; for i:=1 to n do write('*');
```

Пример 10.46. Задано натуральное n (n>2). Напечатать прямоугольный равнобедренный треугольник со стороной катета, равной n, и прямым углом, расположенным справа. Например, для n=5 получим

```
        *
      * *
    * * *
  * * * *
* * * * *
```

Решение. Анализируя структуру ответа, устанавливаем, что нужно напечатать n строк. Каждая строка состоит из двух частей: строки пробелов и строки звездочек. Количество пробелов в начале равно n-1, а затем их количество с каждой строкой уменьшается на единицу. Количество звездочек в начале равно 1, а в каждой новой строке их количество увеличивается на 1.

```
{ фрагмент 47 }
p:=n-1; { начальное количество пробелов }
z:=1; { начальное количество звездочек }
for i:=1 to n do { вывод n строк }
begin
    writeln; { выводим с начала новой строки }
    for j:=1 to p do write(' '); { выводим пробелы }
    p:=p-1; { количество пробелов уменьшается }
    for j:=1 to z do write('*'); { выводим звездочки }
    z:=z+1; { количество звездочек увеличивается }
end.
```

Пример 10.47. Задано натуральное нечетное n (n>3). Напечатать равнобедренный треугольник с вершиной, направленной вверх, и основанием, равным n. Например, для n=5 получим

```
    *
  * * *
* * * * *
```

Решение. Данная задача незначительно отличается от предыдущей. Отличия состоят в том, что нужно печатать (n+1) div 2 строк. Количество звездочек от строки к строке увеличивается на 2. Структура строки осталась прежней. Строка состоит из начальных пробелов (их нужно печатать) и звездочек. Пробелы, завершающие каждую строку, получатся автоматически. Их специально печатать не надо.

```
{ фрагмент 48 }
```

```
p:=(n+1) div 2-1; { начальное количество пробелов }
z:=1; { начальное количество звездочек }
for i:=1 to (n+1) div 2 do { вывод (n+1) div 2 строк }
begin
    writeln; { выводим с начала новой строки }
    for j:=1 to p do write(' '); { выводим пробелы }
    p:=p-1; { количество пробелов уменьшается }
    for j:=1 to z do write('*'); { выводим звездочки }
    z:=z+2; { количество звездочек увеличивается }
end.
```

Упражнения:

1. Задано натуральное n. Напечатать фигуры указанных видов (виды показаны на примере n=5):

```
а)          б)          в)          г)          д)
*           * * * * *   * * * * *       *       * * * * *
* *         * * * *       * * * *     * * *     *       *
* * *       * * *           * * *   * * * * *   *       *
* * * *     * *               * *     * * *     *       *
* * * * *   *                   *       *       * * * * *
```

2. Для произвольного n напечатать фигуру указанного вида:

```
для n=3     для n=5     для n=7
* * *       * * * * *   * * * * * * *
*           *           *
* * *       * * * * *   * * * * * * *
            *           *
            * * * * *   * * * * * * *
                        *
                        * * * * * * *
```

Пример 10.48. Напечатать таблицу функции a·sin(x)+b·cos(x) на отрезке от $-\pi/2$ до $3\pi/2$ с шагом $\pi/18$. Подсчитать количество смен знака в этой таблице.

Решение. Выберем длины выводимых данных (:6:3) и рассчитаем длину печатной строки. В строку входят: стенка (|), пробел, заголовок x, который печатается в шести позициях, так как его длина меньше длины печатаемого значения, пробел, стенка, пробел, заголовок <число> * sin(x) + <число> * cos(x), пробел, стенка. Будем считать, что числа во втором заголовке однозначные. В этом случае длина заголовка 2 (17 символов) больше длины печатаемого значения, поэтому он учтен в длине печатной строки. В результате длина печатной строки получилась равной 40. Структура программы будет такой: вначале программируем вывод шапки, затем в цикле вычисляем значение функции и выводим текущую строку таблицы. Нужно хранить предыдущее значение функции, если произведение текущего значения функции на предыдущее меньше нуля, то функция сменила знак, следовательно, счетчик смен знака увеличиваем на 1.

```
{ фрагмент 49 }
var         a,b:real; { коэффициенты функции }
            x:real; { аргумент }
            y:real; { функция }
            y1:real; { предыдущее значение функции }
            i, { счетчик для печати линии подчеркивания }
            s:integer; { количество смен знака }
            a1,b1:string; { коэффициенты функции в символьном виде }
begin
   write('Введите A и B ');
   readln(a,b);
   { печать шапки }
   for i:=1 to 40 do write('-');writeln; { линия подчеркивания }
   str(a:6:3,a1);str(b:6:3,b1); { коэффициенты в символьном виде }
   writeln('|   x    |  '+a1+'*sin(x)+'+b1+'*cos(x) |');
   { вывели строку заголовков }
   for i:=1 to 40 do write('-');writeln; { линия подчеркивания }
   s:=0; { начальное значение счетчика смен знаков функции }
   x:=-pi/2; { начальное значение аргумента }
   y1:=a*sin(x)+b*cos(x); { предыдущее значение функции }
   while x<=3*pi/2 do { пока не напечатали всю таблицу }
   begin
            y:=a*sin(x)+b*cos(x); { текущее значение функции }
            writeln('| ',x:6:3,' | ',y:6:3,' |'); { вывод строки таблицы }
            x:=x+pi/18; { изменили аргумент на шаг }
            if y*y1<0 then s:=s+1; { если есть смена знака, то увеличиваем
счетчик смен знака функции }
            y1:=y { запомнили текущее значение функции }
   end;
   for i:=1 to 40 do write('-');writeln; { линия подчеркивания }
   writeln('s=',s); { количество смен знака }
end.
```

Пример 10.49. Задано натуральное n и ряд 1+2-3+4+5-6+... . Напечатать вычисляемый ряд, знак равно и ответ, если в ряде используется n слагаемых.

Решение. Подобные задачи уже решались раньше. Данная задача усложняется необходимостью вывода вычисляемого ряда. Это означает, что нужно не только прибавить или вычесть очередное значение, но и вывести его на печать.

```
{ фрагмент 50 }
var         i, { очередной член ряда }
   n, { количество членов ряда }
   s:integer; { сумма }
begin
   write('Введите n ');
   readln(n);
   s:=0; { начальное значение суммы }
   for i:=1 to n do
```

```
    if i mod 3=0 { если очередной член кратен 3, }
    then begin s:=s-i;write('-',i) end { то вычтем его и напечатаем со знаком
минус }
    else   begin s:=s+i; if i=1 { иначе прибавим }
                         then write(i) { если член первый, то знак не
печатаем }
                         else write('+',i) { иначе печатаем плюс }
           end;
    write('=',s);
end.
```

Упражнения:

1. Какую задачу решает представленный ниже фрагмент?

```
{ фрагмент 51 }
var     h1 : integer; { размер строки в позициях }
        h2 : real; { масштаб }
        x : integer; { аргумент }
        y : real; { функция }
        n : integer; { позиция печати функции }
        i : integer; { счетчик позиций }
begin
    writeln('График функции y = |x-2| на интервале [-25;25]');
    h1 := 68;
    h2 := abs(-25-2)/h1;
    for x := -25 to 25 do
    begin  y := abs(x-2);
           n := round(y/h2);
           write('I');
           if x=0
           then   begin  for i:=2 to n-1 do write('-');
                         write('*');
                         for i:=n+1 to h1 do write('-')
                  end
           else   begin  for i:=2 to n-1 do write(' ');
                         write('*')
                  end;
           writeln
    end
end.
```

2. Напечатайте формулу, а затем ответ, полученный вычислением по этой формуле: sin(1+cos(2-sin(3+cos(4+sin(5-cos(6+...)...)). Всего в формуле n пар скобок.

Пример 10.50. Пусть в программу требуется ввести натуральное n, большее 2.

Решение. При вводе обязательно необходима проверка. Если введенное значение не удовлетворяет требуемым условиям, то ввод повторяется. На Паскале решение может быть записано так:

```
repeat
    write('Введите натуральное n>2 ');
```

```
    readln(n)
until n>2.
```

10.10. Переборный метод решения задач

Метод перебора состоит в том, что алгоритм генерирует все возможные комбинации исходных данных задачи и отбирает те из них, которые удовлетворяют заданным условиям. В математике этот метод не приветствуется, к тому же часто прямое применение этого метода приводит к очень большому количеству вариантов, перебрать которые за обозримое время не может даже современный компьютер. Однако применение этого метода в программировании оправдано тем, что для многих задач построить решение с помощью перебора гораздо проще, чем с помощью других методов. В результате очень сложные задачи, решить которые мог только хорошо подготовленный математик, поддаются усилиям человека с очень скромными математическими способностями. Кроме того, можно принять некоторые меры для уменьшения количества перебираемых вариантов за счет исключения заведомо неперспективных.

Пример 10.51. Для каждого четырехзначного числа составляется дробь: отношение суммы цифр числа к самому числу. Найти четырехзначное число, для которого эта дробь наибольшая.

Решение. Например, числу 1234 соответствует дробь 0,0081, а числу 9876 - 0,003. Для перебора всех четырехзначных чисел организуем четыре вложенных цикла. В первом цикле будут перебираться цифры тысяч от 1 до 9 (ноль исключается, потому что число заведомо четырехзначное), во втором - сотни от 0 до 9, в третьем - десятки от 0 до 9, в четвертом цикле - единицы от 0 до 9. Исходным значением искомого отношения может быть ноль.

```
{ фрагмент 52 }
y:=0; { начальное значение отношения }
a:=0; { искомое число }
for i:=1 to 9 do
    for j:=0 to 9 do
            for k:=0 to 9 do
                    for l:=0 to 9 do
                    begin  s:=i+j+k+l;
                            x:=((10*i+j)*10+k)*10+l;
                            if y<s/x then begin y:=s/x; a:=x end
                    end.
```

Пример 10.52. Найти все натуральные трехзначные числа, каждое из которых обладает двумя следующими свойствами:

- первая цифра в три раза меньше последней его цифры;
- сумма самого числа с числом, получающимся из него перестановкой второй и третьей его цифр, делится на 8 без остатка.

Решение. Простое решение можно получить по аналогии с предыдущим:

```
{ фрагмент 53 }
for i:=1 to 9 do { цифра сотен }
    for j:=0 to 9 do { цифра десятков }
            for k:=0 to 9 do { цифра единиц }
```

```
            if (i=3*k) and ((100*i+10*j+k+100*i+10*k+j) mod 8=0)
            then write(i,j,k,' ').
```

Посмотрим, каким образом можно уменьшить перебор. Если цифра сотен в три раза меньше цифры единиц, то k=3*i. Поскольку k - цифра, то k=3*i<=9 или i<=3. Рассмотрим второе условие, приведем в нем подобные члены: (100*i+10*j+3*i+100*i+10*3*i+j) mod 8=0 => (200*i+33*i+11*j) mod 8=0. Поскольку 200*i делится на 8, то его можно отбросить. Таким образом, нужно проверять условие (33*i+11*j) mod 8=0 или 11*(3*i+j) mod 8=0. Получаем следующий фрагмент программы:

```
{ фрагмент 54 }
for i:=1 to 3 do
   for j:=0 to 9 do
            if (3*i+j) mod 8=0
            then write(i,j,3*i).
```

Пример 10.53. Найти все пятизначные числа вида 5m27n (m и n - цифры!), которые делятся на 15.

Решение. Легко построить полный перебор всех возможных случаев.

```
{ фрагмент 55 }
for n:=0 to 9 do
   for m:=0 to 9 do
            if (50000+n*1000+270+m) mod 15=0
            then write(5,n*1000+270+m,' ').
```

Попытаемся уменьшить перебор. Заметим, что число делится на 15, если оно делится на 5 и 3 (15=5·3). Число делится на 5, если оно оканчивается на 0 или 5. Следовательно, значениями m могут быть только цифры 0 или 5. Число делится на 3, если сумма его цифр делится на 3. Три цифры числа известны, их сумма равна 5+2+7=14. Если m=0, то сумма равна 14 и нужно подобрать такое n, чтобы сумма была кратна 3. Отсюда значение n изменяется от 1 до 9 (это цифра!) с шагом 3. Если m=5, то сумма четырех цифр равна 19, следовательно, n изменяется от 2 до 9 с шагом 3. После этих рассуждений перебор свелся к перебору непосредственно решений задачи.

```
{ фрагмент 56 }
n:=1;
while n<=9 do
begin       write(5,n,270,' ');
            n:=n+3
end;
writeln;
n:=2;
while n<=9 do
begin       write(5,n,275,' ');
            n:=n+3
end.
```

Пример 10.54. Последовательность перестановок на множестве {1, 2, ..., n} представлена в лексикографическом порядке, если она записана в порядке возрастания получающихся чисел. Например, лексикографическая

последовательность перестановок трех элементов имеет вид 123, 132, 213, 231, 312, 321. Сформировать все k-элементные перестановки множества {1, 2, 3, ..., n} в лексикографическом порядке.

Решение. Для хранения перестановок будем использовать массив (см. следующую главу).

```
{ фрагмент 57 }
const nn=100;
type mas=array[1..nn] of integer;
var         a:mas;
            i,j,k,n,p:integer;
begin
   write('Введите n и k ');readln(n,k);
   for i:=1 to k do a[i]:=i;
   p:=k;
   while p>=1 do
   begin
         for j:=1 to k do write(a[j],' ');writeln;
         if a[k]=n then p:=p-1 else p:=k;
         if p>=1
         then for i:=k downto p do a[i]:=a[p]+i-p+1;
   end
end.
```

Упражнения:

1. Выполните трассировку фрагмента 57.
2. Установите сколько существует двузначных чисел, сумма квадратов цифр которых делится на 13.
3. Найдите все трехзначные числа, каждое из которых в 19 раз больше суммы своих цифр.
4. Найдите все пятизначные числа вида 67m1n (m и n - цифры!), которые делятся на 36.
5. Сколько слагаемых нужно взять, чтобы в сумме 1+2+3+... получить наименьшее трехзначное число, состоящее из одинаковых цифр?
6. Замените буквы цифрами так, чтобы равенство ab·ck=bbb оказалось верным.
7. Найдите все пятизначные числа, равные кубу числа, образованного двумя их последними цифрами.

Пример 10.55. Любую целочисленную сумму денег большую 7 рублей можно выплатить без сдачи трешками и пятерками. По заданному целому положительному n определить пару целых неотрицательных чисел a и b, таких, что n=3·a+5·b.

Решение. Эта задача решалась нами раньше. Для ее решения было построено два ветвящихся алгоритма, которые выдавали два разных решения. Получим сейчас с помощью перебора все решения этой задачи. Заметим, что значение количества выданных троек не может превысить n div 3, а значение количества

выданных пятерок не может превысить n div 5. Исходя из этого, организуем циклы перебора:

```
{ фрагмент 58 }
repeat
    write('Введите значение n>7 ');
    readln(n);
until n>7;
for a:=0 to n div 3 do
    for b:=0 to n div 5 do
            if n=3*a+5*b
            then writeln('3*',a,'+5*',b,'=',n).
```

Для n=128 получаем следующие решения, причем только два из них могли быть получены ранее с помощью построенных ветвящихся алгоритмов:

3* 1+5*25=128, {решение получено ветвящимся алгоритмом}
3* 6+5*22=128,
3*11+5*19=128,
3*16+5*16=128,
3*21+5*13=128,
3*26+5*10=128,
3*31+5* 7=128,
3*36+5* 4=128,
3*41+5* 1=128. {решение получено ветвящимся алгоритмом}

10.11. Графика в Паскале

В классическом Паскале Н.Вирта не предусмотрены операторы работы с графикой, поэтому рассмотрим этот вопрос в реализации Турбо Паскаля 7.0. Здесь для работы в графическом режиме используется библиотека графических подпрограмм, представленная в виде модуля graph. Настройка графических процедур на работу с конкретным дисплейным адаптером графического режима достигается подключением (инициализацией) нужного графического драйвера, который является специальной программой для управления техническими средствами компьютера, в данном случае графическим экраном. Здесь будем использовать драйвер EGAVGA.BGI для адаптеров EGA и VGA, который находится обычно в каталоге c:\tp7\bgi.

В графическом режиме экран представляет набор пикселов (графических точек), светимостью которых можно управлять, адресуясь к ним с помощью координат.

Пиксел в левом верхнем углу имеет координаты 0,0. Ось x направлена от левого верхнего угла вправо. Максимальная координата по x определяется функцией get-maxx.

Ось y направлена от левого верхнего угла вниз. Максимальная координата по y определяется функцией getmaxy.

Для инициализации (переключения в графический режим) используется последовательность операторов:

```
uses graph;
```

```
var         driver:integer; {используемый драйвер}
            mode :integer; {код графического режима}
            err :integer; {код ошибки}
begin
   driver:=detect;
   initgraph(driver,mode,'c:\tp7\bgi');
   err:=graphresult;
   if err<>0
   then   begin  write('ОШИБКА ГРАФИКИ: ',grapherrormsg(err));
                 halt
          end;
 . . .
closegraph;
end.
```

В графическом режиме используются следующие операторы:

Цвет фона: setbkcolor(цвет). Здесь «цвет» может иметь значение от 0 до 7.

Цвет рисунка: setcolor(цвет). Здесь «цвет» может иметь значение от 0 до 15.

Для «цвета» используется следующая таблица соответствий:

0	Черный
1	Синий
2	Зеленый
3	Голубой
4	Красный
5	Фиолетовый
6	Коричневый
7	Темно-серый
8	Светло-серый
9	Ярко-голубой
10	Ярко-зеленый
11	Ярко-голубой
12	Розовый
13	Ярко-фиолетовый
14	Желтый
15	Белый

Высветить точку с координатами x,y: putpixel(x,y,цвет).

Переместить курсор в новую текущую точку x,y: moveto(x,y).

Прямая линия между точками x1,y1 и x2,y2: line(x1,y1,x2,y2).

Прямая линия от текущей точки до точки x,y: lineto(x,y).

Прямоугольник с левым верхним углом в точке x1,y1 и правым нижним углом в точке x2,y2: rectangle(x1,y1,x2,y2).

Закрашенный прямоугольник с левым верхним углом в точке x1,y1 и правым нижним углом в точке x2,y2: bar(x1,y1,x2,y2).

Установить цвет закраски: setfillstyle(штриховка,цвет).

Штриховка:

0	Цвет фона
1	Сплошная
2	Минусами
3	Косыми чертами
10	Точками

Закрасить область: floodfill(x,y,цвет границы области).

Окружность: circle(x,y,радиус).

Пример 10.56. Построить в заданном прямоугольнике экрана «карту звездного неба» - набор случайных разноцветных точек.

Решение. Исходными данными могут быть либо координаты левого верхнего и правого нижнего углов прямоугольника, либо координата левого верхнего угла прямоугольника и длины его сторон. Кроме того, пользователь должен указать количество звездочек-точек на карте. В этом решении используется первый вариант задания прямоугольника. Для задания координат звездочек-точек используется датчик случайных чисел, попадающих в заданный прямоугольник.

```
{ карта звездного неба }
uses graph;
var     driver:integer; {используемый драйвер}
        mode :integer; {код графического режима}
        err :integer; {код ошибки}
        i :longint; { счетчик цикла }
        x1,y1,x2,y2:integer; { координаты прямоугольника }
        n :integer; { количество точек на карте }
begin
  write('Введите координаты левого верхнего ');
  write('и правого нижнего углов прямоугольника ');
  readln(x1,y1,x2,y2);
  write('Введите количество точек на карте ');
  readln(n);
  { переходим в графический режим }
  driver:=detect;
  initgraph(driver,mode,'c:\tp7\bgi\');
  err:=graphresult;
  if err<>0
  then  begin  write('ОШИБКА ГРАФИКИ: ',grapherrormsg(err));
               halt
        end;
  setbkcolor(0); { цвет фона черный }
  for i:=1 to n do { вывод точек }
        putpixel(x1+random(x2-x1),y1+random(y2-y1),random(15));
closegraph;
end.
```

Пример 10.57. Нарисовать в график функции $y=x^2$ на отрезке от -10 до 10.

Решение. Вспомним процесс построения графика функции в математике. Сначала начертим оси координат и их подпишем. Выберем расположение осей координат так, чтобы они пересекались в центре экрана. Поскольку мы перенесли оси координат и их перевернули по сравнению с исходным положением, то поэтому координаты рисуемых на экране точек будут вычисляться по формулам: x=x0+абсцисса, y=y0-ордината. Здесь x0,y0 - координаты центра новой системы координат.

Выберем затем масштаб по осям x и y. Масштаб показывает, какая часть экрана изображает единицу реального графика. В тетради по математике часто выбирается масштаб, при котором одна клеточка тетради равна единице реального графика. Чтобы задать масштаб на экране нужно выбрать размеры экрана, отводимые для вывода графика. Пусть в нашем случае по оси x используется количество точек getmaxx для изображения отрезка [-10;10] реального графика, поэтому mx=getmaxx/(10-(-10)+1).

По оси y для изображения отрезка $[0;10^2]$ будем использовать getmaxy div 2 точек, поэтому my=getmaxy/100.

Выбрав масштаб, нанесем единицы измерения на построенные оси. Поскольку на оси y много точек, будем наносить их с шагом 10.

Построив оси, строим график, нанося отдельные точки и соединяя их отрезками прямых (строим график непрерывной функции!!) так, чтобы получилась плавная кривая. Иногда, для достижения плавности, нужно увеличить количество точек. Построение в нашем случае графика, состоящего из большого количество точек, неприемлемо, так как функция не точечная, а непрерывная.

```
uses graph;
var     d,m :integer; { для инициализации графики }
        x0,y0:integer; { координаты центра осей координат графика }
        x,y,i:integer; { текущая точка и счетчик циклов }
        mx,my:real; { масштабы }
        s :string; { для разметки осей }
begin
   d:=detect;
   initgraph(d,m,'c:\tp7\bgi');
   setbkcolor(7);
   setcolor(1);
   x0:=getmaxx div 2;
   y0:=getmaxy div 2;
   { Рисуем оси координат }
        { ось X }
   line(0,y0,getmaxx,y0);
   line(getmaxx,y0,getmaxx-10,y0-10);
   line(getmaxx,y0,getmaxx-10,y0+10);
   outtextxy(getmaxx-10,y0-15,'X');
        { ось Y }
   line(x0,0,x0,getmaxy);
   line(x0,0,x0-10,10);
   line(x0,0,x0+10,10);
```

```
    outtextxy(x0-12,5,'Y');
            { масштабы }
    mx:=getmaxx/(10-(-10)+1);
    my:=getmaxy div 2/100;
            { наносим единицы измерения на оси }
    for i:=-10 to 10 do
    begin  x:=x0+round(i*mx);
            line(x,y0-2,x,y0+2);
            str(i,s);
            outtextxy(x+1,y0+2,s);
    end;
    for i:=0 to 100 do
    begin  y:=y0-round(i*my);
            if (i mod 10=0) and (i<>0)
            then    begin  line(x0-2,y,x0+2,y);
                           str(i,s);
                           outtextxy(x0+4,y,s);
                    end
    end;
     { вывод графика }
    x:=x0+round(-10*mx);
    y:=y0-round(100*my);
    moveto(x,y);
    for i:=-9 to 10 do
    begin  x:=x0+round(i*mx);
            y:=y0-round(i*i*my);
            lineto(x,y);
    end;
  closegraph;
  end.
```

Упражнения:

1. Измените программу так, чтобы она строила график $y=2 \cdot x^2$.
2. Постройте графики функций, рассмотренные в главе о ветвящихся алгоритмах.
3. Постройте картинку: домик, забор, трава и солнце.

11. МЕТОДЫ РАБОТЫ С МАССИВАМИ

11.1. Понятие массива

Для скалярных (простых) данных (переменных и констант) характерно то, что их значение занимает одну ячейку памяти, хотя это и может составить несколько байт. Обращение (чтение) к хранимому значению ведется по имени. Читать это значение можно сколько угодно раз, но при записи нового значения старое безвозвратно теряется. Простыми такие данные называются потому, что их внутреннее устройство (строение, структура) простое: одно данное - одна ячейка памяти.

Наряду с простыми имеются данные, обладающие сложной структурой (устройством, строением). Данные сложной структуры занимают несколько ячеек памяти. Самыми распространенными данными сложной структуры являются массивы.

Массив объединяет несколько однородных (принадлежащих одному типу) данных в единую структуру. Данные, объединяемые в массив, называются элементами. Каждому элементу отводится своя ячейка памяти. Элементы одного массива занимают последовательно расположенные ячейки. Все элементы массива имеют одно общее имя - имя массива, и один общий тип значений - тип элементов. Количество элементов в массиве называют его размером. Размер массива играет большую роль в момент резервирования (отведения места) памяти для элементов массива. Для того чтобы отвести место в памяти для элементов массива, их количество должно быть известно. Если отведение места в памяти осуществляется в момент трансляции и при исполнении алгоритма не изменяется, то такое распределение памяти называется статическим, а соответствующие им массивы - статическими. Такое распределение памяти просто реализовать при разработке трансляторов, поэтому оно широко распространено. В Паскале массивы статические.

Возможно динамическое распределение памяти, когда отведение места элементам происходит в момент выполнения программы (динамические массивы). В Паскале для их реализации используются указатели. Далее, если не оговорено противное, рассматриваются статические массивы. Для того чтобы отыскать информацию об отдельном элементе массива, с каждым из них связывается некоторое значение (несколько значений), называемое индексом (индексами). Комбинация имени массива и индекса (индексов) однозначно определяет элемент массива. Индекс указывает расположение конкретного элемента массива по отношению к его началу. Если в качестве значений индекса используется начальный отрезок натурального ряда, то индекс совпадает с номером элемента в массиве. Если для обращения к элементу массива используется один индекс, то массив называется одномерным. Его можно рассматривать как аналог (модель) вектора в математике. Если для обращения к элементу массива используется два индекса, массив называется двумерным. Его можно рассматривать как аналог (модель) матрицы в математике. Языки программирования не запрещают использовать больше двух индексов для обращения к элементу массива. Однако в практике программирования это применяется редко, потому что любой алгоритм удается запрограммировать, используя только одномерные и/или двумерные массивы.

На рис. 11.1 показан массив A, состоящий из семи элементов, каждый из которых - целое число. Здесь представлены три варианта индексов: целые числа от -3 до 3, натуральные числа от 1 до 7, символы от a до g.

Номер элемента:	1	2	3	4	5	6	7
М А С С И В A:	-7	13	2	-40	52	2	-3

Индексы:	1)	-3	-2	-1	0	1	2	3
	2)	1	2	3	4	5	6	7
	3)	a	b	c	d	e	f	g

Рис. 11.1. Одномерный массив и возможные значения его индексов

Структуры, аналогичные массивам, достаточно широко распространены в обыденной жизни. Например, в качестве массива можно рассматривать всех учеников некоторого класса, пассажиров парохода или самолета, производственную бригаду и т. д. Все эти коллективы имеют одно общее имя. Обратиться к отдельному члену коллектива можно, например, указав его порядковый номер в списке состава. Книга тоже массив. Она состоит из страниц, к каждой из которых можно обратиться, назвав книгу и номер страницы. Кресла в кинотеатре так же массив. Найти нужное кресло можно, указав название кинотеатра, номер ряда и номер кресла в ряду.

Преимуществом объединения в массив является то, что для указаний операций над всем массивом не нужно указывать эту операцию для каждого элемента в отдельности. Например, по объявлению «На субботник выходят школьники 7б класса» выходит весь класс. Такое объявление дать легче, чем перечислить всех по отдельности учеников 7б класса.

Использование индекса позволяет осуществить прямой доступ к элементам массива. Прямой доступ означает, что для обращения к i-му элементу нет необходимости просматривать предыдущие и последующие элементы. Значения индекса могут быть представлены выражением, что делает массивы одним из мощных средств программирования.

11.2. Массивы в Паскале

Перед использованием массивы в Паскале нужно описать. Описание задает тип индексов и тип элементов. Для одномерного массива оно имеет вид: *type имя_нового_типа = array[тип1] of тип2*; где тип 1 - скалярный или ограниченный тип индекса (целый и вещественный типы не допускаются); тип 2 - любой тип, разрешенный в Паскале. Для двумерного массива описание имеет вид:

type имя_нового_массива=array[тип1] of array [тип2] of тип_элементов;

Имеется сокращенная форма описания двумерного массива:

type имя_нового_массива=array[тип1,тип2] of тип_элементов;

По описанию массива в момент трансляции программы резервируется место в памяти для элементов массива. Резервируемое количество элементов для одномерного массива определяется как количество значений типа, соответствующего индексу. Например, если тип индекса - диапазон: *нижняя_граница .. верхняя_граница*, то количество элементов вычисляется по формуле: верхняя_граница - нижняя_граница+1. Для двумерного массива количество элементов определяется перемножением количеств элементов соответствующих типов. Если тип 1 и тип 2 - диапазоны, то количество элементов в массиве равно: *(верхняя_граница1 - нижняя_граница1) * (верхняя_граница2 - нижняя_граница2)*.

Примеры типов:

1. type mas=array [1..10] of integer; Определяет тип массива, состоящего из 10-1+1=10 элементов целого типа. Индекс задается диапазоном целых чисел от 1 до 10.

Var x, y, z : mas; Определяет переменные x, y, z типа mas - одномерные массивы. Эти массивы относятся к одному типу.

2. Type mas1=array ['a'..'z',boolean] of real; Определяет тип массива, состоящего из 26·2=52 элементов вещественного типа. Первый индекс (индекс строк) - диапазон строчных латинских букв от 'a' до 'z'. Второй индекс (индекс столбцов) - логические значения от false до true.

Var p,q,s,r:mas1; Определяет переменные p,q,s,r типа mas1. Эти массивы также относятся к одному типу.

3.
```
const n=12;
type    vector = array [1..n] of real;
        matrix1 = array [1..n] of array [1..n] of boolean;
        matrix2 = array [1..3,1..6] of integer;
        mas2 = array [1..2*n] of vector;
        days = (mon, tue, wed, thu, fri, sat, san);
var     a : vector;
        b : array [-7..12] of char;
        c : matrix1;
        d : matrix2;
        e,f : mas2;
        g,h : array [mon..fri] of string;
```

В разделе описания типов описаны следующие типы данных: одномерный массив vector, состоящий из n (12) элементов типа real; двумерный массив matrix1, состоящий из n·n (144) логических элементов и содержащий n строк и n столбцов; двумерный массив matrix2, состоящий из 18 элементов целого типа и содержащий 3 строки и 6 столбцов; одномерный массив mas2, состоящий из 2·n (24) элементов типа vector, этот же массив можно рассматривать как двумерный, содержащий 2·n строк и n столбцов, каждый элемент которого имеет тип real (всего 288 элементов); перечислимый тип days состоит из перечислений названий дней недели.

Обратите внимание, что вместо того, чтобы задавать размерность некоторых массивов числом 12, вначале определена константа n, равная 12. Этот простой прием позволяет при изменении количества элементов в массиве модифицировать только одну строку программы n=12.

В разделе описаний переменных объявлены следующие переменные: массив a типа vector; одномерный массив b, содержащий 12-(- 7)+1=20 элементов типа char; массив c типа matrix1; массив d типа matrix2; массивы e и f типа mas2 и одномерные массивы g и h, содержащие по 5 элементов типа string, индексы этого массива принадлежат перечислимому типу days.

Поскольку массивы в Паскале статические, то количество элементов в них должно быть известно в момент трансляции. Если необходимо использование динамических массивов, то приходится описывать массивы с максимально возможным в данной задаче количеством элементов, а затем использовать только часть из этих элементов. Это неудобно для многих задач. Выйти из этого

положения можно с помощью динамических массивов, для организации которых используются указатели.

Чтобы обратиться в тексте программы к конкретному элементу массива, необходимо указать имя массива, а затем в квадратных скобках через запятую перечислить индексы элемента. Здесь индексы могут быть заданы выражением соответствующего типа. При использовании выражений в качестве индекса приходится следить за тем, чтобы значение индекса не вышло за объявленные пределы. Поскольку не все компиляторы Паскаля отслеживают эту ситуацию, следить за ней приходится программисту, причем тщательно, так как выход значения индексного выражения за объявленные границы является очень распространенной ошибкой.

Рассмотрим примеры обращений к элементам описанных ранее массивов. К элементу массива x[2*j-1] можно обратиться, если значение индексного выражения (2*j-1) находится на отрезке от 1 до 10. Также можно обратиться к элементам массивов y и z.

Запись p['y',false] позволит обратиться к элементу массива p, находящемуся в строке с индексом 'y' и столбце с индексом false. Также можно обратиться к элементам массивов q, r и s.

К третьему элементу массива a можно обратиться так: a[3]. Обращение a[2*i] будет правильным, если 2*i будет иметь значение от 1 до n (12).

К элементам массива b можно обратиться так: b[-3], b[0], b[10]. Обращение b[i] будет верным, если значение i изменяется от -7 до 12.

К элементам массива d можно обратиться так: d[i,j]. Здесь первый индекс указывает номер строки, а второй - столбца. Первый индекс может изменяться от 1 до 3, а второй - от 1 до 6.

К элементам массивов e или f можно обратиться как e[3][7] или e[3,7] (f[3][7] или f[3,7]), причем вторая запись является сокращением первой. К третьей строке этих массивов, как к одномерным массивам, можно обратиться так: e[3] (f[3]). Первый индекс изменяется от 1 до 2·n (24), а второй - от 1 до n (12).

К элементу массивов g и h можно обратиться так: g[wed] (h[wed]), где значение индекса изменяется от mon до fri.

Над массивами одного типа в Паскале можно выполнять операцию присваивания. При этом первый элемент массива, указанного в правой части оператора присваивания, переписывается в первый элемент массива, указанного в левой части оператора, второй - во второй элемент, третий - в третий и т.д. Массивы считаются однотипными, если имеют тип с одним и тем же именем или перечислены через запятую в строке, где описаны они сами и их типы. В примерах однотипными являются массивы: x,y,z; p,q,r,s; e,f; g,h. Следовательно, возможно присваивание y:=z, что равносильно следующей последовательности операторов присваивания: y[1]:=z[1]; y[2]:=z[2]; y[3]:=z[3]; y[4]:=z[4]; y[5]:=z[5]; y[6]:=z[6]; y[7]:=z[7]; y[8]:=z[8]; y[9]:=z[9]; y[10]:=z[10].

Возможно также присваивание a:=e[4], что вызовет перенос четвертой строки массива e в массив a. Эта операция аналогична операции переименования некоторых объектов. Например, по окончанию учебного года 7a класс переименовывается в 8a. Это означает, что первый по списку ученик 7a станет первым по списку учеником 8a, второй - вторым, третий - третьим и т.д.

Остальные операции над массивами выполняются поэлементно. Перед рассмотрением этих операций остановимся на ошибках, характерных при работе с массивами:

1. Имя массива ошибочно используется в качестве имени другого данного в программе.

2. Обращение к элементу массива осуществляется со значением индекса, выходящим за его границы. Эта ошибка встречается наиболее часто.

3. Указанные при описании массива в качестве верхней или нижней границы индекса выражения могут быть вычислены только во время исполнения программы, поэтому транслятор не может зарезервировать память для элементов массива (точно также нельзя заказать автобусы для экскурсии, если неизвестно (хотя бы приблизительно) количество желающих принять в ней участие).

Использование массива в программе не всегда оправдано, потому что обращение к элементу массива требует вначале вычисления значения индекса (индексов). Только после этого осуществляется доступ собственно к элементу. Имеются, по крайней мере, два случая когда необходимо применение массива:

1. когда все данные должны быть представлены в памяти для обработки и любое данное может потребоваться в любой момент (классическим примером является задача сравнения всех элементов некоторой последовательности с ее последним элементом);

2. когда результаты вычислений или промежуточные значения, содержащиеся в нескольких ячейках памяти, должны быть сохранены в памяти и все они несут одинаковую «логическую функцию». Эти ячейки могут быть сгруппированы под одним именем и образуют массив.

Пусть, например, необходимо подсчитать количество вхождений всех букв русского алфавита в некоторый текст. Для решения задачи можно ввести 33 счетчика - по одному для каждой буквы алфавита. Все 33 счетчика выполняют одну и ту же логическую функцию. Их можно сгруппировать в массив, присвоив массиву имя СЧЕТЧИК, содержащий 33 элемента. Индекс его изменяется от 1 до 33, что позволяет различать счетчики: СЧЕТЧИК(1) содержит количество вхождений «А», СЧЕТЧИК(2) - количество вхождений «Б» и т.д. СЧЕТЧИК(33) содержит количество вхождений «Я».

Рассмотрим пример работы с массивом как со структурой с прямым методом доступа.

Пример 11.1. Вычислить сумму нескольких произвольных элементов массива, индексы которых вводятся с клавиатуры. Признаком окончания ввода является ноль.

В этом и следующих примерах будем считать, что описан одномерный массив с нижней границей индекса 1 и верхней - nn. В массиве используются только n элементов.

```
Const nn=50;
type mas = array [1..nn] of real;
var         a : mas; {исходный массив}
            n : integer; { количество используемых элементов <=nn}
            i,j,k : integer; { индексы массива }
```

Решение. Пусть, например, n=5 и массив a содержит следующие значения: a[1]=-3, a[2]=17, a[3]=4, a[4]=-5, a[5]=81. Последовательность ввода имеет вид: 3, 1, 10, 3, 5, 0. В этом случае искомая сумма вычисляется в следующем порядке:

1) s:=0;
2) s:=s+a[3] = 0+4 = 4;
3) s:=s+a[1] = 4+(-3)=1;
4) сообщение пользователю: «В массиве нет элемента с индексом 10»;
5) s:=s+a[3] = 1+4=5;
6) s:=s+a[5] = 5+81=86
7) 0 - конец ввода.

В результате получена сумма s=86.

Для решения задачи введем элементы массива. Затем организуем цикл ввода элементов до того момента, когда будет введен ноль. В теле этого цикла будем проверять допустимость индекса и находить сумму или выдавать сообщение о неправильном индексе. Фрагмент этого алгоритма на Паскале будет выглядеть так:

```
{Ввод массива}
write('Введите используемое количество элементов ');readln(n);
write('Введите ',n,' элементов массива ');
for i:=1 to n do read(a[i]);
{Вычисление суммы}
s:=0;
write('Введите индекс ');read(i);
while i<>o do
begin     if (1<=i) and (i<=n)
          then s:=s+a[i]
          else writeln('В массиве нет элемента с индексом ',i);
          write('Введите индекс ');read(i);
end;
{Вывод ответа} writeln('s=',s)
```

11.3. Способы перебора элементов массива

Часто при работе с массивами задача ставится так, что требуется все элементы или их часть обработать одинаково. Для такой обработки организуется перебор элементов. Схему перебора элементов массива можно охарактеризовать:

а) направлением перебора;
б) количеством одновременно обрабатываемых элементов;
в) характером изменения индекса.

По направлению перебора различают схемы:

- от первого элемента к последнему (от начала массива к концу);
- от последнего элемента к первому (от конца к началу);
- от обоих концов к середине.

В дальнейшем первый элемент одномерного массива будем называть его началом, а последний - концом.

В массиве одновременно можно обрабатывать один, два, три и т.д. элемента, причем, если обрабатывается больше одного элемента, то возможна обработка соседями или группами по n.

Рассмотрим, например, перебор соседями по два. Каждый элемент массива, кроме первого и последнего, имеет двух соседей, по одному слева и справа. Первый элемент имеет одного соседа справа, а последний элемент - одного соседа слева. Здесь можно осуществлять перебор так, что обрабатывается каждый раз текущий элемент со своим соседом справа. Например, если задан массив a[1], a[2],a[3], a[4], a[5], a[6], то при описанном выше алгоритме элементы будут перебираться такими парами: a[1] и a[2], a[2] и a[3], a[3] и a[4], a[4] и a[5], a[5] и a[6].

При переборе группами по n обработанные n элементов массива повторно не обрабатываются. Например, при переборе массива a[1], a[2], a[3], a[4], a[5], a[6] группами по два элемента обработка будет проведена в следующей последовательности: a[1] и a[2], a[3] и a[4], a[5] и a[6].

При переборе элементов индекс может меняться в арифметической, геометрической прогрессии. Характер изменения индекса может быть и более сложным - нелинейным.

В самом общем случае схема перебора элементов имеет вид:

параметр_цикла := начальное значение, попадающее в заданный диапазон индекса массива; .

ЦИКЛ ПЕРЕБОРА пока выполняется условие продолжения цикла:

обработка выбранных элементов массива;

изменение индекса в соответствии с выбранной формулой изменения: параметр_цикла := F(параметр_цикла);

проверка попадания нового значения параметра цикла в диапазон изменения индекса;

КОНЕЦ цикла перебора.

Часто в качестве параметра цикла используется индекс массива. Обратите внимание также на то обстоятельство, что после изменения индекса его необходимо сразу же проверить на попадание в заданный диапазон, иначе возможны ошибки.

Рассмотрим конкретные схемы перебора.

Случай 1. Перебрать элементы массива по одному, двигаясь от начала массива к концу. Здесь индекс начального элемента 1, индекс последнего обрабатываемого элемента n, шаг перебора 1. Конечное значение (кз) параметра цикла при условии проверки окончания с помощью сравнения <= может быть вычислено по формуле:

(конечное значение - начальное значение + 1)/шаг = n.

Отсюда: (кз-1+1)/1=n или кз=n.

Схема перебора может быть представлена в виде

```
for i:=1 to n do {обработка a[i]}
```

или

```
i:=1;
while i<=n do
begin      {обработка a[i]}
           i:=i+1
end.
```

Если условие окончания проверяется с помощью сравнения <, то конечное значение вычисляется так: (конечное значение - начальное значение)/шаг=n.

Отсюда: (кз - 1)/1=n или кз=n+1. Схема перебора может быть представлена в виде

```
i:=1;
while i<n+1 do
begin      { обработка a[i]}
           i:=i+1
end.
```

Случай 2. Перебрать элементы массива по одному, двигаясь от конца массива к началу.

```
for i:=n downto 1 do {обработка a[i]}
или
i:=n;
while i>=1 do
begin      {обработка a[i]}
           i:=i-1
end.
```

Случай 3. Обработать массив по одному элементу, двигаясь с обоих концов к середине массива.

```
i:=1; {установка нижней границы}
j:=n; {установка верхней границы}
while i<=j do
{ если обработан элемент a[i],
 то обработать a[j]; j:=j+1;
 иначе обработать a[i]; i:=i+1 }
```

Анализируя рассмотренные схемы перебора, можно заметить, что в правильно построенной схеме обязательно должны присутствовать блок установки начальных значений индексов массива, блок проверки индекса (индекс не должен выходить за границы индексов массива), блок изменения индекса для перехода к следующему элементу массива, причем, за блоком изменения индекса по времени выполнения должен располагаться блок проверки индекса на принадлежность интервалу, определенному границами массива. Если будет нарушено хотя бы одно из перечисленных условий, то в процессе выполнения программы возникнут ошибки.

Рассмотрим пример. Пусть схема перебора элементов массива по одному запрограммирована следующим фрагментом:

```
i:=1;
while i<=n do
begin      i:=i+1;
           {обработка a[i]}
end.
```

В этом фрагменте элемент a[1] не будет обработан, кроме этого будет осуществлена попытка обратиться к элементу массива a[n+1], а так как такого элемента не существует, то это приведет к ошибке. В данном примере нарушено требование обязательной проверки индекса после его изменения. Для исправления ошибок заметим, чтобы индекс попадал в интервал от 1 до n, условие окончания цикла должно проверять индекс на попадание в интервал от 0 до n-1. Тогда фрагмент с исправлениями будет иметь вид

```
i:=0;
```

```
while i<n do
begin     i:=i+1;
          {обработка a[i]}
end.
```

Рассмотрим еще один пример. Пусть задача предыдущего примера реализована следующим фрагментом:

```
i:=1;
while i<=n do
          {обработка a[i]};
          i:=i+1;
```

При попытке исполнения приведенного фрагмента компьютер зациклится. Здесь нарушено условие, требующее обязательного изменения индекса массива после обработки очередного элемента. Ошибка возникла потому, что по правилам Паскаля телом оператора цикла является только один, следующий за ключевым словом do оператор (до ближайшей точки с запятой). Если в тело цикла входит более одного оператора, то из них необходимо образовать составной оператор, объединив несколько операторов в один с помощью операторных скобок begin end. После исправления получим фрагмент, совпадающий с первым случаем.

Случай 4. Перебрать элементы массива с четными индексами, двигаясь от начала к концу.

Приведем три варианта построения схемы.

Вариант 1. Здесь индекс начинает изменяться с четного числа, величина шага, равная двум, обеспечивает сохранение четности индекса.

```
i:=2;
while i<=n do
begin     {обработка a[i]}
          i:=i+2
end.
```

Вариант 2. Здесь внутрь цикла перебора вложен оператор, проверяющий четность индекса.

```
for i:=1 to n do
   if i mod 2=0
   then {обработка a[i]}.
```

Работает эта схема дольше предыдущей, но иногда она необходима.

Вариант 3. Здесь для получения элемента с четным индексом используется формула четного числа. Поскольку элементов с четным индексом половина от общего количества, то переменная цикла i изменяется от 1 до n div 2 (при целочисленном делении дробная часть отбрасывается).

```
for i:=1 to n div 2 do
   {обработка a[i]}
```

Случай 5. Перебрать элементы массива с четными индексами, двигаясь от конца массива к его началу.

Здесь установка начального значения - не простое присваивание, а условный оператор, позволяющий отыскать последний элемент массива с четным индексом.

```
if n mod 2 = 0 then i:=n else i:=n-1;
while i>0 do
```

```
begin     {обработка a[i]}
          i:=i-2
end.
```

Условный оператор, устанавливающий начальное значение индекса, можно внести в тело цикла, тогда получим следующий вариант схемы:

```
i:=n;                                    или   for i:=n downto 1 do
while i>0 do                                       if i mod 2 = 0
begin     if i mod 2 = 0                       then {обработка a[i]}
          then {обработка a[i]};
          i:=i-1
end.
```

Он медленнее приведенного выше, так как предполагает проверку четности индекса при каждом исполнении цикла.

Случай 6. Перебрать элементы массива с четным индексом, двигаясь с обоих концов массива к его середине.

Для решения этой задачи соединим схемы перебора, рассмотренные в случах 4 и 5:

```
i:=2;
if n mod 2 = 0 then j:=n else j:=n-1;
while i<=j do
begin     {если обработан элемент a[i],
          то обработать a[j]; j:=j-2;
          иначе обработать a[i]; i:=i+2 }
end.
```

Случаи 4-6 можно легко обобщить для перебора любых элементов, кратных k. Рассмотрим схему перебора элементов массива, кратных k, двигаясь от начала массива к его концу.

```
i:=k;
while i<=n do
begin     {обработка a[i]}
          i:=i+k
end.
```

Перебор элементов массива, кратных k, можно осуществить следующим образом:

```
if n mod k = 0 then i:=n
else if n mod k = 1 then i:=n-1
else if n mod k = 2 then i:=n-2
else if n mod k = 3 then i:=n-3
else i:=n-4;
while i>0 do
begin     {обработка a[i]}
          i:=i-k
end.
```

Поскольку при формировании начального значения i, из n вычитается величина остатка от деления на k, то вложенные условные операторы можно заменить оператором присваивания: i:=n-n mod k;

Рассмотрим схемы перебора нескольких элементов массива одновременно.

Случай 7. Перебрать соседние элементы массива, двигаясь от начала массива к концу. Рассмотрим вначале случай двух соседей. Для массива из 5 элементов нужно последовательно обработать пары: a[1] и a[2] , a[2] и a[3], a[3] и a[4], a[4] и a[5].

Вариант 1. for i:=1 to n-1 do {обработать a[i] и a[i+1]}.

Вариант 2. for i:=2 to n do {обработать a[i-1] и a[i]}.

Обобщим эту схему на случай k соседей. Для массива из 5 элементов и k=3 нужно последовательно обработать тройки: a[1]-a[2]-a[3], a[2]-a[3]-a[4], a[3]-a[4]-a[5]. Здесь возможны три варианта реализации в зависимости от того, какой элемент из тройки выбрать текущим:

1) a[i] - a[i+1] - a[i+2], в этом случае перебор идет от 1 до n-2;

2) a[i-1] - a[i] - a[i+1], в этом случае перебор идет от 2 до n-1;

3) a[i-2] - a[i-1] - a[i], в этом случае перебор идет от 3 до n.

Для k элементов возможны k вариантов схем перебора в зависимости от выбора текущего элемента. Пример, когда текущим является первый элемент для каждых k соседей , приведен ниже:

{n-(k-1) означает, что у последних k-1 элемента нет k соседей }

for i:=1 to n-k+1 do {обработка a[i], a[i+1],...,a[i+k-1}.

В следующем примере приводится та же схема перебора, текущим является последний элемент для каждых k соседей:

for i:=k to n do {у первых k-1 элементов нет k соседей справа }

{ обработка a[i-k+1], a[i-k+2],...,a[i] }.

Случай 8. Перебрать соседние элементы массива, двигаясь от конца массива к его началу. Сначала рассмотрим схему для двух соседей. В случае массива, состоящего из 5 элементов, обработка должна проводиться в таком порядке: a[5] и a[4], a[4] и a[3], a[3] и a[2], a[2] и a[1]. Здесь можно так же, как и в предыдущем случае, предложить два варианта построения схемы.

Вариант 1. for i:=n downto 2 do {обработка a[i] и a[i-1]}.

Вариант 2. for i:=n-1 downto 1 do {обработка a[i] a[i+1] }.

Можно обобщить эту схему на случай k соседей. Здесь возможны k различных вариантов реализации схемы в зависимости от того, какой элемент среди k соседей выбран текущим. Ниже приводятся варианты схем для случаев, когда текущим элементом является первый и последний.

Вариант 1. for i:=n downto k do {обработка a[i-k+1],...,a[i-1],a[i]}.

Вариант 2. for i:=n-k+1 downto 1 do {обработка a[i],a[i+1],...,a[i+k-1]}.

Случай 9. Перебрать все элементы массива группами по k, двигаясь от начала к концу. Например, для массива, состоящего из пяти элементов, обработка группами по два (k=2) должна вестись так: a[1] и a[2], a[3] и a[4]. У элемента a[5] нет пары для образования группы, поэтому он не обрабатывается.

Обработка этого же массива группами по три должна вестись так: a[1], a[2] и a[3]. Элементы a[4] и a[5] группы по три не образуют.

Из рассмотренных примеров видно, что в этой схеме перебора шаг равен размеру группы (k), а индекс последнего элемента, который может быть обработан, не должен быть больше, чем n-k+1. Однако проверку окончания можно ослабить, заменив ее на проверку невыхода индекса за верхнюю границу, так как перебор ведется с шагом k и элементы, не образующие группы, будут просто пропущены. Здесь также возможны различные варианты в зависимости от выбора текущего элемента в группе. Всего таких вариантов k. Ниже приводятся варианты, в которых текущий элемент выбран первым и последним.

Вариант 1.

```
i:=1;
while i<=n-k+1 do
begin
    {обработка a[i],a[i+1],...,a[i+k-1]}
    i:=i+k
end.
```

Вариант 2.

```
i:=k;
while i<=n do
begin
    {обработка a[i-k+1],a[i-k+2],...,a[i]}
    i:=i+k
end.
```

Упражнения:

1. Сравнить схему перебора 9 со схемами перебора 4-6, установить сходства и различия.

2. Записать схему перебора 9 для k=8 во всех вариантах (их должно быть 8).

Случай 10. Перебрать все элементы массива группами по k, двигаясь от конца массива к его началу.

Один из возможных вариантов перебора следующий:

```
i:=n;
while i>=k do
begin
    {обработка a[i-k+1],a[i-k+2],...,a[i]}
    i:=i-k
end.
```

11.4. Перебор подмассивов

В программировании часто возникают задачи, в которых для заданного элемента массива нужно перебрать оставшиеся элементы. Например, для каждого элемента перебрать все ему предшествующие или последующие элементы, перебрать элементы от заданного до первого элемента, удовлетворяющего некоторому условию. Для перебора каждого подмассива можно использовать рассмотренные ранее схемы. Это приводит к необходимости вводить такое количество индексов, каково количество обрабатываемых подмассивов. Рассмотрим конкретные примеры.

Пример 11.2. Для каждого элемента массива просмотреть все его последующие элементы. Здесь возможно несколько вариантов.

Вариант 1. Массив просматриваем по одному элементу слева направо. Подмассив просматриваем по одному тоже слева направо. Обозначим i - индекс массива, а j - индекс подмассива, тогда схема может быть такой:

```
for i:=1 to n-1 do {у последнего элемента нет последующего}
   for j:=i+1 to n do  {обработка a[j]}
```

Вариант 2. Подмассив просматриваем справа налево.

```
for i:=1 to n-1 do
begin
   j:=n;
   while j>i do
   begin
           {обработка a[j]}
           j:=j-1
   end
end.
```

Могут быть использованы и схемы перебора парами, соседями и т.д.

Пример 11.3. Для каждого элемента просмотреть все ему предшествующие. Пусть i - индекс массива, а j - индекс подмассива.

Случай 1. Основной массив просматривается слева направо, подмассив просматривается также слева направо.

```
for i:=2 to n do {у первого элемента нет предыдущего}
   for j:=1 to i-1 do
           {обработка a[j]}
```

Случай 2. Основной массив просматривается слева направо, а подмассив - справа налево.

```
for i:=2 to n do
begin
   j:=i-1;
   while j>0 do
   begin
           {обработка a[j]}
           j:=j-1
   end
end.
```

Случай 3. Основной массив просматривается справа налево, а подмассив - слева направо.

```
i:=n;
while i>1 do
begin
   for j:=1 to i-1 do {обработка a[j]};
   i:=i-1
end.
```

Случай 4. Основной массив просматривается справа налево, а подмассив - справа налево.

```
i:=1;
while i>1 do
begin
    j:=i-1;
    while j>0 do
    begin
            {обработка a[j]}
            j:=j-1
    end;
    i:=i-1
end.
```

11.5. Нелинейные схемы перебора элементов массива

Нетрудно заметить, что все рассмотренные ранее схемы перебора основывались на линейном изменении индекса массива, т.е. один шаг цикла отличался от другого шага цикла на величину постоянную и неизменную. Если величина шага цикла будет величиной переменной, то имеем случай нелинейной схемы перебора.

Пример 11.4. Нужно обработать элементы массива с индексами 2, 4, 8, 16, 32, ... Получаем нелинейную схему перебора:

```
i:=2;
while i<=n do
begin
    {обработка a[i]}
    i:=i*2
end.
```

В общем случае функция изменения индекса может быть очень сложной. При ее реализации может потребоваться организация дополнительного цикла для изменения шага основного цикла или организация специальной процедуры/функции, вычисляющей значение шага цикла.

Рассмотрим еще пример. Пусть необходимо обработать элементы массива в следующей последовательности: сначала - все элементы с нечетными индексами, затем элементы четные, делящиеся на 2, но не на 4, затем все делящиеся на 4, но не на 8 и т.д.

Выпишем требуемую последовательность индексов для n = 10 (количество элементов в массиве равно 10): 1, 3, 5, 7, 9, 2, 6, 10, 4, 8. Анализируя полученную последовательность индексов, замечаем, что для ее получения массив должен быть просмотрен несколько раз. Во время одного просмотра шаг не меняется, а при переходе к следующему просмотру шаг увеличивается вдвое. Поэтому для такой схемы перебора необходимо организовать два цикла. В первом (внешнем) происходит изменение шага, а во втором (внутреннем) - просмотр массива с выбранным шагом. Индекс начала просмотра определяется значением шага предыдущего просмотра массива. Обозначим h - шаг просмотра; j - индекс элемента, с которого начинается просмотр; k - индекс обрабатываемого элемента, тогда схему перебора можно записать так:

```
h:=1; { начало просмотра должно начинаться с элемента, принадлежащего
данному массиву }
```

```
j:=h;
while j<=n do
begin
    h:=h*2;
    k:=j;
    while k<=n do
    begin
            {обработка a[k]}
            k:=k+h
    end;
    j:=h
end;
```

Упражнение. Проведите трассировку представленной выше схемы перебора.

В приведенном решении использовалась дополнительная переменная j для указания индекса начала просмотра и переменная h изменялась в начале тела цикла. Для устранения этих «недостатков» предлагается другое решение. Количество просмотров массива можно определить по формуле $z=]\log_2 n[+1$. Здесь n - количество элементов в массиве,]x[- функция, находящая целую часть x, т.е. наибольшее целое, не превосходящее данное. Эта формула была получена следующим образом. Для случая n=10 приведенный алгоритм выполняет четыре просмотра. В первом просмотре обрабатываются индексы 1,3,5,7,9, шаг просмотра равен 2, во втором просмотре - индексы 2,6,10, шаг просмотра равен 4, в третьем просмотре - индекс 4, шаг просмотра равен 8, в пятом просмотре - индекс 8, шаг просмотра равен 16. Можно заметить, что шаг на каждом просмотре увеличивается в два раза, следовательно, количество просмотров (z) конкретного массива, зависящее от количества элементов в нем, можно определить из неравенств $2^{z-1}<n\le 2^z$. Отсюда $z=]\log_2 n[+1$, а максимальное значение $h=2^{]\log_2 n[+1}$. На основе этого можно переписать предыдущую схему нелинейного перебора элементов массива, сведя ее к двум вложенным циклам. В первом цикле изменяется h, а во втором - индекс массива k.

```
h:=2;
while h<=exp((trunc(ln(n)/ln(2))+1)*ln(2)) do
begin
    k:=h div 2;
    while k<=n do
    begin
            {обработка a[k]}
            k:=k+h
    end;
    h:=h*2
end.
```

В данном примере свели нелинейное изменение индекса массива к изменению двух переменных по закону арифметической прогрессии.

Пример 11.5. Напечатать те элементы одномерного массива, которые являются полными квадратами. Здесь для вычисления индекса можно использовать две переменные. P - индекс элемента массива, i - номер квадрата. Известен квадрат

натурального числа 1, он равен 1, тогда фрагмент нужной программы может быть таким:

```
i:=1;
p:=sqr(i);
while p<=n do
begin
    write(a[p]);
    i:=i+1;
    p:=sqr(i)
end.
```

Пример 11.6. Напечатать те элементы одномерного массива, индексы которых являются числами Фибоначчи. Здесь для вычисления следующего значения индекса нужно хранить два предыдущих, поэтому для вычисления значения индекса будем использовать три переменных: a - текущее значение числа Фибоначчи, b - предыдущее значение числа Фибоначчи, c - предпредыдущее значение числа Фибоначчи.

```
c:=1;
b:=1;
while b<= n do
begin
    write(a[b]);
    a:=b+c;
    c:=b;
    b:=a
end.
```

11.6. Классы задач по обработке массивов

Все задачи по обработке массивов можно разбить на перечисленные ниже классы, причем, отнесение задачи к тому или иному классу определяет выбор возможных методов решения задачи. Эти классы относятся как к одномерным, так и к двумерным массивам. Здесь рассматривается применение вводимой классификации для одномерных массивов. Двумерным массивам будет посвящена отдельная глава.

Классы задач:

1) однотипная обработка всех или указанных элементов массива;

2) задачи, в результате решений которых изменяется структура (порядок следования элементов) массива;

3) обработка нескольких массивов одновременно. Сюда же относятся задачи, когда по-разному обрабатываются подмассивы одного и того же массива;

4) поисковые задачи для массивов.

11.6.1. Решение задач первого класса

Решение задач первого класса сводится к установлению того, как обрабатывается каждый элемент массива или указанные элементы. Затем выбирается подходящая схема перебора элементов, в которую вставляются операторы обработки элементов массива.

Пример 11.7. Найти сумму (произведение) элементов одномерного массива.

Решение. Каждый элемент массива прибавляется к сумме s:=s+a[i] или умножается на произведение предыдущих элементов массива p:=p*a[i]. Поскольку требуется перебрать все элементы массива, можно выбрать схему перебора элементов по одному, двигаясь с начала или с конца массива.

```
{сумма элементов массива}  {произведение элементов массива}
s:=0;                      p:=1;
i:=1;                      i:=n;
while i<=n do              while i>0 do
begin                      begin
   s:=s+a[i];                      p:=p*a[i];
   i:=i+1                          i:=i-1
end.                       end.
```

Пример 11.8. Подсчитать количество элементов массива, совпадающих по величине с заданным.

Решение. В этой задаче исходными данными являются одномерный массив и некоторое число, которое обозначим X. Для ответа на вопрос задачи нужно каждый элемент массива сравнить с заданным. Если они совпадут, то счетчик совпавших элементов нужно увеличить на единицу. Поскольку нужно сравнить каждый элемент, то нужно выбрать схему перебора по одному элементу. В приведенном здесь решении выбрана схема перебора по одному от последнего элемента к первому. Ответ задачи будет храниться в счетчике совпавших элементов. Исходя из приведенных рассуждений, можно предложить такой вариант решения:

```
s:=0;
for i:=n downto 1 do
   if a[i]=x then s:=s+1.
```

Пример 11.9. Найти максимальный элемент одномерного массива.

Решение. В этой задаче «действующие лица» - одномерный массив и его максимальный элемент. Одномерный массив является исходным данным, а максимальный элемент - результатом. Максимальный элемент массива будем хранить в переменной max. Для его поиска достаточно каждый элемент массива сравнить с max (max<a[i]). Начальным значением максимального элемента может быть значение произвольного элемента массива, чаще всего для этой цели выбирают первый элемент. Исходя из этих соображений, получаем следующий фрагмент программы:

```
const nn=100;
type mas=array [1..nn]of integer;
var        a:mas;   {исходный массив}
           n:integer; {количество используемых элементов}
           i:integer; {индекс массива}
           max:integer; {максимальный элемент массива}
begin
   max:=a[1];
   for i:=2 to n do
           if max<a[i] then max:=a[i];
   writeln('максимальный элемент массива ',max);
```

end.

Упражнения:

1. Выполните трассировку фрагмента программы примера 11.9. Что получим в ответе, если все элементы массива будут равны?

2. Как изменится работа программы, если оператор цикла будет заменен на for i:=1 to n do? Какой из этих двух фрагментов будет работать быстрее? Почему?

3. Напишите фрагмент программы, в которой оператор в теле цикла остался таким же, а схема просмотра перебирала бы элементы от последнего к первому.

4. Пусть массив содержит несколько элементов, совпадающих по величине с максимальным. Установите, какой по счету максимальный элемент найдет приведенный в примере фрагмент алгоритма. Как изменится работа программы, если в цикле заменить условие на max<=a[i]?

5. Напишите фрагменты программ для поиска минимального элемента в одномерном массиве.

Попытаемся ускорить работу алгоритма поиска максимального элемента одномерного массива. Для этого выберем схему просмотра массива по два элемента. Тогда на каждом шаге цикла нужно будет выбрать максимальный элемент из трех: max, a[i],a[i+1]. Соответствующий фрагмент алгоритма приводится ниже:

```
const nn=100;
type mas=array [1..nn]of integer;
var
   a:mas;   {исходный массив}
   n:integer; {количество используемых элементов}
   i:integer; {индекс массива}
   max:integer; {максимальный элемент массива}
begin
   max:=a[1];
   i:=1;
   while i<=n do
   begin
         if a[i]<a[i+1]
         then   begin
                     if max<a[i+1]
                     then max:=a[i+1]
                end
         else   begin
                     if max<a[i]
                     then max:=a[i]
                end;
         i:=i+2
   end;
   if n mod 2=1
   then   if max<a[n]
         then max:=a[n];
   writeln('максимальный элемент массива ',max);
```

```
end.
```

В данном фрагменте шаг цикла равен двум, поэтому массив будет просмотрен в два раза быстрее, чем при схеме перебора по одному элементу.

Упражнение. Примените для решения данной задачи схему перебора элементов по три. Какие выводы можно сделать относительно скорости работы полученной программы и предыдущих программ? Обобщите ваши выводы на случай перебора по m предметов.

Пример 11.10. Найти номер максимального элемента массива.

Решение. В задаче используются понятия массива и номера максимального элемента. Обозначим массив так же, как в предыдущих примерах: номер текущего элемента массива будем обозначать i, а номер максимального элемента - j. Как обычно, для определения максимального элемента нужно сравнить каждый элемент с максимальным. В выбранных обозначениях такое сравнение запишется так: if a[j]<a[i] then j:=i. Поскольку нужно просмотреть все элементы массива, то, выбрав подходящую схему перебора, например, перебор по одному от первого элемента к последнему, получим следующее решение задачи. Здесь начальное значение номера максимального элемента выбрано равным 1, но оно может быть любым от 1 до n.

```
j:=1;
for i:=2 to n do
    if a[j]<a[i] then j:=i;
write('номер максимального элемента ',j,' максимальный элемент ',a[i]).
```

Упражнения:

1. Сравните фрагменты программ поиска максимального элемента и поиска номера максимального элемента для одинаковых схем перебора. Чем они отличаются?

2. Что делает фрагмент приведенной ниже программы?

```
max:=a[1]; j:=1;
for i:=2 to n do
    if a[j]<a[i] then begin max:=a[i]; j:=i; end;
write('номер максимального элемента ',j,' максимальный элемент ',a[i]).
```

Эффективна ли она по сравнению с программой предыдущего примера?

3. Напишите программу, которая находит номер первого (последнего) из нескольких элементов, совпадающих по величине с максимальным.

Другие решения задачи можно получить, если выбрать другие схемы перебора элементов в массиве. Выберем, например, схему перебора элементов по одному, двигаясь от обоих концов массива к середине. Обозначим через i номера начальных элементов массива, а через j - номера последних элементов массива. Для поиска максимального элемента нужно сравнить все элементы между собой. В данном случае будем сравнивать i-й и j-й элементы и изменять значение индекса для того элемента, который оказался меньше. По окончании просмотра оба индекса будут указывать на один и тот же элемент, он и будет являться максимальным.

```
i:=1; j:=n;
while i<j do
    if a[i]<a[j]
    then i:=i+1
```

```
    else j:=j-1;
write('максимальный элемент ',a[i],' номер максимального элемента ',j).
```

Упражнения:

1. Выполните трассировку приведенного фрагмента для случая, когда все элементы массива различны и когда все элементы массива равны.

2. Как, используя эту же схему перебора, найти первый (последний) совпадающий по величине с максимальным элемент, если в массиве есть несколько таких элементов.

Пример 11.11. Подсчитать количество элементов массива, совпадающих по величине с максимальным.

Решение. Одним из очевидных решений будет сведение этой задачи к уже решенным. Найти максимальный элемент можно с помощью алгоритма, приведенного в примере 11.9. Затем, воспользовавшись алгоритмом примера 11.8, можно подсчитать количество элементов массива, совпадающих по величине с максимальным. Такой подход требует двойного просмотра массива. Его реализация приведена ниже:

```
max:=a[1];
for i:=2 to n do
    if max<a[i] then max:=a[i];
s:=0;
for i:=1 to n do
    if a[i]=max then s:=s+1;
```

Сравнив два цикла в приведенном решении, заметим, что их можно объединить в один, увеличив скорость работы алгоритма.

```
max:=a[1]; s:=1; { один совпавший с текущим максимальным уже есть, это
a[1]}
for i:=2 to n do
    if max<a[i]
    then    begin {найден новый кандидат на максимум}
                    max:=a[i];
                    s:=1
            end
    else if max=a[i] then s:=s+1.
```

Пример 11.12. Подсчитать максимальное количество нулевых элементов одномерного массива, расположенных подряд.

Решение. В этой задаче используется одномерный массив, являющийся исходной информацией. Обозначим максимальное количество нулевых элементов через m. Имеющееся количество нулевых элементов в текущей последовательности будем считать с помощью переменной s. Будем просматривать массив поэлементно. Если выбранный элемент является нулем, то счетчик s увеличиваем на 1. Если выбранный элемент не нуль, то счетчик s обнуляем. Затем сравниваем m и s и максимальное из них записываем в m. Воспользуемся схемой перебора по одному от первого к последнему элементу, тогда получим следующий алгоритм:

```
{пример 11.12, вариант 1}
m:=0; {нулей пока не обнаружено}
s:=0;
```

```
for i:=1 to n do
begin     if a[i]=0
          then s:=s+1
          else s:=0;
          if m<s then m:=s
end;
write('максимальное количество нулей, следующих подряд ',m).
```

Для поиска ошибок в предложенном алгоритме необходимо разработать набор тестов, чтобы каждая ветка алгоритма была выполнена хотя бы по разу. Предлагается следующий набор тестов:

1. n=9, a={1,2,3,4,5,6,7,8,9}. Ответ: максимальное количество нулей, следующих подряд 0. Начало последовательности 0.
2. n=9, a={0,0,0,0,0,0,0,0,0}. Ответ: максимальное количество нулей, следующих подряд 9. Начало последовательности 1.
3. n=9, a={0,1,0,0,2,0,0,0,3}. Ответ: максимальное количество нулей, следующих подряд 3. Начало последовательности 6.
4. n=9, a={1,0,2,0,0,3,0,0,0}. Ответ: максимальное количество нулей, следующих подряд 3. Начало последовательности 7.
5. n=9, a={1,0,1,0,1,0,1,0,1}. Ответ: максимальное количество нулей, следующих подряд 1. Начало последовательности 2.
6. n=9, a={3,0,0,0,2,0,0,1,0}. Ответ: максимальное количество нулей, следующих подряд 3. Начало последовательности 2.

Упражнение. Проведите трассировку для каждого из приведенных тестов.

Исправим алгоритм так, чтобы он указывал номер позиции, с которой начинается максимальная последовательность нулей. В тот момент, когда обнаруживается максимальное значение s, индекс i указывает на последний элемент последовательности подряд идущих нулей, s хранит количество нулей в этой последовательности. Отсюда, номер первого элемента можно определить по формуле i+1-s или, поскольку m и s равны, i+1-m. Обозначим номер первого элемента в последовательности подряд идущих нулей через j. Соответствующий фрагмент алгоритма приведен ниже:

```
{пример 11.12, вариант 1 с определением j}
m:=0; {нулей пока не обнаружено}
s:=0;
j:=0;
for i:=1 to n do
begin     if a[i]=0
          then s:=s+1
          else s:=0;
          if m<s then begin m:=s; j:=i+1-m end
end;
write('максимальное количество нулей, следующих подряд ',m,' ',j).
```

Анализ первого варианта алгоритма показывает, что нет необходимости проверять значение s на максимум после каждого элемента массива. Это достаточно делать только для ненулевых элементов. Для этого нужно перенести второй условный оператор в ветку else первого условного оператора. Кроме того,

нужно вынести этот же оператор за цикл, чтобы проверить m и s, когда массив заканчивается нулем. В новом алгоритме проверка осуществляется, когда i указывает на первый ненулевой элемент после последовательности нулей, поэтому номер начала последовательности нулей определится по формуле j=i-m или для последнего элемента j=n+1-m.

```
{пример 11.12, вариант 2 с определением j}
m:=0; {нулей пока не обнаружено}
s:=0;
j:=0;
for i:=1 to n do
    if a[i]=0
    then s:=s+1
    else    begin if m<s then begin m:=s;j:=i-m end;
                  s:=0
            end;
{массив может закончиться нулем, поэтому нужно проверить на максимум}
    if m<s then begin m:=s;j:=n+1-m end;
    write('максимальное количество нулей, следующих подряд ',m,' ',j).
```

Анализируя второй вариант алгоритма, можно заметить, что проверка на максимум будет осуществляться для каждого ненулевого элемента, однако логика задачи подсказывает, что такая проверка должна проводиться только по окончании последовательности подряд идущих нулей. Заметим, что конец последовательности можно определить, анализируя два рядом стоящих символа. Если первый символ нуль, а рядом справа стоит не нуль, то это и будет признак конца последовательности. Это правило определения конца последовательности не работает для массива, оканчивающегося на нуль. Чтобы исправить этот недостаток, добавим в конец массива один фиктивный ненулевой элемент. В этом случае для решения задачи нужна схема перебора массива по два элемента. Соответствующий алгоритм приводится ниже:

```
{пример 11.12, вариант 3 с определением j}
m:=0; {нулей пока не обнаружено}
s:=0;
j:=0;
a[n+1]:=1; { дописали в массив фиктивный элемент для универсального
определения окончания последовательности рядом стоящих нулей }
i:=1;
while i<n+1 do
begin
    if (a[i]=0) and (a[i+1]<>0)
    then    begin
                s:=s+1;
                if m<s then   begin  m:=s;
                                     s:=0;
                                     j:=i+1-m
                              end
                else s:=0
```

```
            end
    else if a[i]=0 then s:=s+1;
    i:=i+1
end;
write('максимальное количество нулей, следующих подряд ',m,' ',j).
```

Можно предложить следующий вариант решения этой задачи: дополнить массив фиктивным ненулевым элементом, просматривать его по одному элементу. Ненулевые элементы не будут обрабатываться. Для нулевых организуем вложенный цикл подсчета подряд следующих нулей. Как только цикл завершится, а это всегда произойдет, так как в конце массива добавлен ненулевой фиктивный элемент, проверим подсчитанное количество элементов на максимум. Номер начала последовательности можно запомнить перед циклом подсчета нулей. Соответствующий вариант программы приведен ниже:

```
{пример 11.12, вариант 4 с определением j}
m:=0; {нулей пока не обнаружено}
j:=0;
a[n+1]:=1; { дописали в массив фиктивный элемент для универсального
определения окончания последовательности рядом стоящих нулей }
i:=1;
while i<n+1 do
begin   if a[i]=0
        then    begin  s:=1; {первый нуль найден!}
                       j1:=i; {запомнили начало последовательности нулей}
                       i:=i+1;
                       while a[i]=0 do
                       begin  s:=s+1;
                              i:=i+1
                       end;
                       if m<s then begin m:=s;j:=j1 end;
                end;
        i:=i+1
end;
write('максимальное количество нулей, следующих подряд ',m,' ',j).
```

Четвертый вариант решения относится к третьему классу задач на массивы, так как последовательность нулей можно рассматривать как подмассив. О методах решения таких задач рассказывается дальше.

Пример 11.13. Подсчитать максимальное количество следующих подряд совпадающих элементов.

Решение. Например, в массиве A={1,2,2,3,3,3,4,4,4,4} больше всего четверок. Начинается эта последовательность с 7-го элемента. В массиве B = {4, 2, 4, 4, 3, 4, 4, 4, 5, 4, 4} также больше всего четверок. Наиболее длинная последовательность из подряд идущих четверок содержит их 3 штуки и начинается с 6-го элемента.

Данная задача очень похожа на предыдущую, поэтому для ее решения можно использовать те же методы. Для проверки совпадения двух соседних элементов массива можно использовать условие a[i-1]=a[i]. Вариант решения, основанный на подходе варианта 4 примера 11.12, приводится ниже:

```
    m:=0;
    j:=0;
    a[n+1]:=a[n]+1; {нужно, чтобы фиктивный элемент не совпал с последним}
i:=1;
    while i<n+1 do
    begin
       s:=1; { считаем, что a[i] является первым элементом интересующей нас
последовательности }
       j1:=i; {запомнили номер ее начала}
       i:=i+1;
       while a[i-1]=a[i] do {проверяем, совпадают ли соседние элементы}
       begin  s:=s+1;
              i:=i+1
       end;
       if m<s then begin m:=s;j:=j1 end; {проверка на максимум}
    end;
    write('максимальное количество рядом расположенных совпавших ',m,' ',j).
```

Упражнения:

1. Постройте систему тестов и проверьте на ней работу приведенного выше варианта алгоритма.

2. Постройте алгоритмы решения задачи пример 11.13, используя другие подходы примера 11.12 или собственные идеи.

3. Приведите формулировки других задач, которые можно решить, используя подходы примера 11.12.

Пример 11.14. Вычислить сумму произведений элементов заданного одномерного массива: $x_1x_2+x_2x_3+...+x_{n-1}x_n$.

Решение. Здесь требуется найти сумму произведений элементов массива. Всего элементов в сумме n-1. Каждое произведение состоит из двух сомножителей: индекс первого сомножителя совпадает с номером слагаемого, а индекс второго сомножителя на единицу больше. Здесь используем схему перебора по два.

```
s:=0;
for i:=1 to n-1 do s:=s+a[i]*a[i+1];
```

Пример 11.15. Вычислить сумму произведений элементов заданного одномерного массива: $x_1x_n+x_2x_{n-1}+x_3x_{n-2}+...+x_nx_1$.

Решение. Здесь, как и в предыдущем примере, требуется найти сумму n слагаемых, каждое из которых представляет произведение двух сомножителей. Индекс первого сомножителя совпадает с номером слагаемого, а индекс второго сомножителя можно найти из следующих соображений. Заметим, что сумма индексов элементов массива, составляющих каждое произведение, равна n+1: 1+n=n+1, 2+n-1=n+1, 3+n-2=n+1, ..., n+1=n+1. Тогда, зная номер одного сомножителя (i), можно найти номер второго сомножителя j=n+1-i.

```
s:=0;
for i:=1 to n do s:=s+a[i]*a[n+1-i];
```

Пример 11.16. Вычислить сумму квадратов элементов с нечетными индексами заданного одномерного массива.

Решение. Здесь находится сумма элементов с нечетными индексами. Нечетный индекс можно получить либо начав с некоторого нечетного числа, а затем прибавляя к нему два, либо воспользоваться формулой нечетного числа. Решения, основанные на этих подходах, приведены ниже:

```
{Решение 1. Перебор с шагом два }
s:=0;
i:=1;
while i<=n do
begin     s:=s+a[i]*a[i];
          i:=i+2
end;

{Решение 2. Используем формулу нечетного числа }
s:=0;
i:=1;
while 2*i-1<=n do
begin     s:=s+a[2*i-1]*a[2*i-1];
          i:=i+1
end;
```

Пример 11.17. Вычислить произведение сумм элементов заданного одномерного массива: $x_n(x_n+x_{n-1})(x_n+x_{n-1}+x_{n-2})...(x_n+x_{n-1}+...+x_1)$.

Решение. Здесь нужно найти произведение n сомножителей. Каждый сомножитель представляет собой сумму элементов массива. Каждая сумма отличается от предыдущей добавлением одного элемента массива. Просмотр массива здесь лучше всего вести с конца.

```
p:=1;
s:=0;
for i:=n downto 1 do
begin     s:=s+a[i];
          p:=p*s
end;
```

Пример 11.18 Заполнить массив случайными трехзначными целыми числами.

Решение. «Случайно выбранные» числа широко используются в информатике, например, при моделировании разных явлений (приход клиентов в банк, выбор респондентов при опросе общественного мнения, тасование карт, бросание игральных костей), в численном анализе (метод Монте - Карло), для получения данных для тестирования, для принятия решений (каждый, наверное, хотя бы однажды принимал решение бросая монету) и т.д. В Паскале есть специальная стандартная функция random, значением которой является случайное число из интервала [0;1). Эта функция обычно называется датчиком случайных чисел. При каждом вызове функции это число каждый раз свое. Если же вызывать функцию много раз подряд, то множество полученных чисел будет равномерно распределено в интервале [0;1). При повторном запуске программы последовательность случайных чисел повторяется. Это сделано для удобства отладки программ. Таким образом, числа эти получаются с помощью специального алгоритма, реализуемого функцией random, поэтому их еще называют псевдослучайными.

Функция random(a) возвращает целое псевдослучайное число из интервала [0;a). Если нужно случайное число из интервала [a;b], то применяют выражение a+random(b+1-a). Например, для получения случайной цифры - выражение random(10). Для получения случайного числа из интервала [-60;50] используют выражение -60+random(51-(-60)) = -60+random(111). Применяя описанное, пример 11.18 можно на Паскале записать так:

```
for i:=1 to n do a[i]:=10+random(90).
```

11.6.2. Решение задач второго класса

Характерным признаком задач второго класса является изменение порядка следования элементов массива. Для этого часто приходится менять местами элементы массива. Эта работа в Паскале выполняется с помощью приведенных ниже операторов:

```
{обменять местами элементы a[i] и a[j]}
r:=a[i];     { запомнили a[i] во вспомогательной переменной }
a[i]:=a[j]; { записали значение a[j] на место a[i], старое значение
последнего потерялось, но сохранилась его копия в
переменной r }
a[j]:=r      { записали запомненное значение a[i] на место a[j] }
```

Частой ошибкой при решении задач этого класса является потеря нужного значения вследствие записи на его место другого значения. Ниже приводятся характерные задачи этого класса и методы их решений.

Пример 11.19. Разделить каждый элемент одномерного массива на его пятый элемент.

Решение. Например, для массива a={1, 2, 3, 4, 5, 6, 7, 8, 9} должен получиться результат a={1/5, 2/5, 3/5, 4/5, 1, 6/5, 7/5, 8/5, 9/5}. Ясно, что пятый элемент результата всегда будет равен 1, поэтому исходное значение этого элемента нужно сохранить, иначе последующие элементы будут вычислены неправильно. Для хранения исходного значения пятого элемента используем переменную r. Поскольку требуется разделить КАЖДЫЙ элемент массива, то выберем схему перебора по одному. Вообще, если в условии задачи встречаются слова КАЖДЫЙ, ВСЯКИЙ, ДЛЯ ВСЕХ элементов массива, то это означает, что в решении появится цикл перебора элементов. Таким образом, формулировка задачи может подсказать решение.

```
r:=a[5];
for i:=1 to n do a[i]:=a[i]/r;
```

Пример 11.20. Задан одномерный массив. Нужно переставить его элементы в обратном порядке. Другой массив использовать не разрешается. Например, для массива a={1,2,3,4,5,6,7,8,9} получится результат a={9,8,7,6,5,4,3,2,1}.

Решение. Заметим, что при преобразовании меняются местами два элемента: первый и последний, второй и предпоследний и так далее. В случае нечетного количества элементов в массиве имеется центральный элемент, который должен меняться сам с собой, т.е. остаться на месте. Меняются элементы, расположенные симметрично относительно центра массива. Индексы каждой пары таких элементов можно определить по следующим формулам: i и n+1-i. Всего в массиве должно произойти n div 2 обменов. Последняя фраза легко переводится на Паскаль: нужно

организовать арифметический цикл со счетчиком до n div 2, в теле которого меняются местами элементы a[i] и a[n+1-i].

```
for i:=1 to n div 2 do
begin   r:=a[i];
        a[i]:=a[n+1-i];
        a[n+1-i]:=r
end;
```

Другое решение этой задачи можно получить, если ввести обозначения для пары меняемых элементов: i - элемент, входящий в пару и расположенный в начале массива, а j - соответствующий элемент пары, расположенный в конце массива. При таком соглашении i должно быть меньше j. Если i=j, то они указывают на один и тот же элемент, который можно не менять. Если i>j, то пар больше нет. Таким образом, имеем итерационный цикл, который выполняется до тех пор, пока есть пары элементов. В теле цикла a[i] и a[j] меняются местами.

```
i:=1; j:=n;
while i<j do
begin   r:=a[i];
        a[i]:=a[j];
        a[j]:=r;
        i:=i+1; j:=j-1
end.
```

Пример 11.21. Задан одномерный массив. Нужно переставить в обратном порядке его элементы, начиная с элемента с индексом p и заканчивая элементом с индексом q. Дополнительный массив использовать не разрешается. Например, для массива a={1,1,2,3,4,5,6,7,7,7,7,8} и p=3, а q=7 получится результат a={1,1,6,5,4,3,2,7,7,7,7,8}.

Решение. Заметим, что эта задача похожа на предыдущую. В ней изменялся массив целиком, а здесь только часть, расположенная между элементами с индексами p и q включительно. Решение этой задачи легко получается из второго решения предыдущей задачи заменой начальных значений переменных i и j.

```
i:=p;
j:=q;
while i<j do
begin   r:=a[i];
        a[i]:=a[j];
        a[j]:=r;
        i:=i+1;
        j:=j-1
end.
```

Упражнения:

1. Решите задачу примера 11.21, используя идею решения примера 11.20. Какое решение легче для понимания? Почему?

2. Как Вы думаете, какая задача является более общей: задача примера 11.20 или задача примера 11.21? Почему?

Пример 11.22. Сдвинуть элементы одномерного массива на r элементов вправо. Выпадающие из массива элементы становятся в его начало. Например, для

исходного массива a={1,2,3,4,5,6,7,8,9} и r=3 последовательно получаем: после первого сдвига a={9,1,2,3,4,5,6,7,8}, после второго сдвига a={8,9,1,2,3,4,5,6,7}, после третьего сдвига a={7,8,9,1,2,3,4,5,6}.

Решение. Описанную выше идею можно положить в основу алгоритма. Тогда для решения задачи нужно r раз выполнить следующее: запомнить последний элемент; сдвинуть все оставшиеся элементы на один вправо; записать запомненный элемент на первое место в массиве. Получаем решение:

```
for i:=1 to r do
begin     s:=a[n];
          for j:=n-1 downto 1 do a[j+1]:=a[j];
          a[1]:=s
end.
```

Упражнение. Напишите аналогичный алгоритм для сдвига элементов одномерного массива на L влево.

Недостатком предложенного алгоритма является его медленная работа. В самом деле, для сдвига массива на r элементов требуется просмотреть массив r раз. Если воспользоваться алгоритмом переворота части массива, то можно получить результат за один просмотр массива. Для этого последовательно выполняем следующие операции (операции показаны на примере массива a={1,2,3,4,5,6,7,8,9} и r=3):

1) переворачиваем часть массива между индексами 7 - 9; получаем: a={1,2,3,4,5,6,9,8,7};

2) переворачиваем часть массива между индексами 1 - 6; получаем: a={6,5,4,3,2,1,9,8,7};

3) переворачиваем весь массив; получаем: a={7,8,9,1,2,3,4,5,6}.

Упражнения:

1. Запишите описанный алгоритм на Паскале.

2. Докажите, что в этом алгоритме массив просматривается один раз при любом r.

3. Напишите аналогичный алгоритм для сдвига массива влево.

4. Утверждается, что приведенная ниже программа сдвигает на L элементов влево одномерный массив a, состоящий из n элементов. Проведите трассировку и проверьте правильность утверждения.

```
x:=L;
y:=n;
while x<>y do
    if x<y then y:=y-x else x:=x-y;
for i:=1 to x do
begin     this:=i;
          temp:=a[this];
          next:=this+L;
          if next>n then next:=next-n;
          while next<>i do
          begin  a[this]:=a[next];
                 this:=next;
                 next:=next+L;
```

```
            if next>n then next:=next-n
        end;
        a[this]:=temp
end.
```

11.6.3. Решение задач третьего класса

В общем случае, когда обрабатываются несколько массивов одновременно, для каждого массива нужно выбрать подходящую схему перебора, завести свой индекс, следить, чтобы индекс не вышел за границы массива. В некоторых частных случаях для обработки нескольких массивов бывает достаточно одного индекса, потому что элементы массива обрабатываются «синхронно», т.е., зная индекс элемента одного массива, можно вычислить по некоторой формуле индекс соответствующего ему элемента другого массива. Если такой формулы установить не удается, то говорят, что массивы обрабатываются «асинхронно».

Рассмотрим примеры типичных алгоритмов решения задач этого класса.

Пример 11.23. Заданы два одномерных массива, содержащие координаты двух векторов в n-мерном пространстве. Нужно найти координаты вектора суммы.

Решение. Из математики известно, что координаты вектора-суммы находятся как суммы соответствующих координат векторов слагаемых. Здесь первая координата первого вектора складывается с первой координатой второго вектора и получается первая координата суммы, вторая со второй и т.д., поэтому здесь несмотря на то, что в работе участвуют три массива, достаточно одного индекса. В фрагменте алгоритма используются следующие обозначения: a,d - массивы координат заданных векторов, c - массив координат вектора-суммы, i - общий индекс массива, n - размерность векторов.

```
for i:=1 to n do c[i]:=a[i]+b[i];
```

Пример 11.24. Заданы два массива. В первом хранятся некоторые числа, второй содержит только нули и единицы. Нужно в третий массив на те же самые места переписать те элементы первого массива, которым во втором массиве соответствуют единицы. Например, для исходных массивов a и b получится результат c. Минус обозначает неопределенное значение.

```
a={  2  3  4  5  6  7  8   9}
b={  0  0  1  0  1  1  0   1}
c={  -  -  4  -  6  7  -   9}
```

Решение. Здесь массивы также обрабатываются «синхронно», поэтому достаточно одного индекса.

```
for i:=1 to n do
    if b[i]=1 then c[i]:=a[i];
```

Пример 11.25. Массив a длины n перенести в массив b так, чтобы первый элемент массива a стал k-м элементом массива b. Предполагается, что массив a целиком поместится в массиве b.

Решение. В этой задаче легко установить правило, в соответствии с которым по известному индексу массива a можно установить соответствующий индекс массива b. Здесь i-й элемент массива a соответствует i+k-1-му элементу массива b. Для решения задачи будем перебирать по одному элементы исходного массива

(массива a), вычислять им соответствующие места в массиве b и переписывать элементы. Такое решение назовем «циклом по исходному массиву».

```
for i:=1 to n do b[i+k-1]:=a[i];
```

Можно поступить по-другому. Перебирать места в массиве b и вычислять индексы элементов массива a, которые должны быть расположены на данных местах. Назовем такое решение «циклом по выходному массиву». Соответствующая формула здесь такова: на j-м месте в массиве b должен быть помещен j-k+1-й элемент массива a.

```
for j:=k to n+k-1 do b[j]:=a[j-k+1];
```

Упражнение. Совершите перенос массива a так, чтобы его последний элемент, т.е. n-й элемент, стал L-м элементом массива b. Предполагается, что массив a целиком поместится в массиве b. Задачу решите двумя способами: циклом по исходному массиву и циклом по выходному массиву.

Пример 11.26. Переписать из массива a в массив b положительные элементы, прижав их в массиве b к его началу.

Решение. Здесь исходным массивом является массива a, выходным - b. Просматриваем поэлементно исходный массив и, обнаружив положительный элемент, переписываем его в массив b. Здесь нельзя заранее установить формулу, по которой на основе индекса элемента массива a вычисляется соответствующий индекс в массиве b. Задача относится к «асинхронным». Введем разные индексы для массива a (i) и массива b (j). Тогда задача может быть решена так:

```
j:=0; {в массиве b нет пока ни одного элемента}
for i:=1 to n do
   if a[i]>0 then begin
                          j:=j+1; {готовим место для следующего элемента}
                          b[j]:=a[i]
                 end.
```

Заметим, что в конце работы предложенного алгоритма значение j будет равно количеству элементов в массиве b.

Упражнение. Выполните трассировку программы, расположенной ниже. Какую работу она выполняет? Сравните данное решение с предложенным выше, укажите преимущества и недостатки каждого решения.

```
j:=1;
for i:=1 to n do
   if a[i]>0 then begin b[j]:=a[i]; j:=j+1 end.
```

Пример 11.27. Из массива a получить массив b, в котором элементы массива a расположены в таком порядке: $a_n a_1 a_{n-1} a_2 a_{n-2} a_3 \ldots$.

Решение. Легко заметить, что элементы в массиве b чередуются следующим образом. На первом месте стоит последний элемент из массива a, на втором - первый, на третьем - предпоследний. Таким образом, на нечетных местах располагаются элементы с конца массива a в порядке убывания индексов. На четных местах в массиве b располагаются элементы с начала массива a в порядке возрастания индексов. Для этого решения выбираем схему перебора по элементам массива b (цикл по выходному массиву). Элементы массива a перебираем по схеме « обоих концов к середине» Запишем это решение на Паскале:

```
i:=1; {индексы массива a}
```

```
j:=n;
for k:=1 to n do
   if odd(k)
   then begin b[k]:=a[j]; j:=j-1 end
   else begin b[k]:=a[i]; i:=i+1 end.
```

Упражнение. Установите какую работу выполняет приведенная ниже программа. Сравните это решение с приведенным выше. Почему в данном решении по окончании цикла необходим условный оператор?

```
i:=1;
j:=n;
for k:=1 to n div 2 do
begin     b[2*k-1]:=a[j];
          b[2*k]:=a[i];
          j:=j-1;
          i:=i+1
end;
if odd(n) then b[n]:=a[j].
```

Можно решить пример 11.27, перебирая элементы исходного массива. В этом случае индекс выходного массива будет изменяться с шагом 2. Сначала этот индекс должен быть четным и возрастать, так как начальные элементы массива a располагаются на четных местах в порядке возрастания. После того, как индекс выходного массива превысит n, он должен принять нечетное значение и уменьшаться с шагом 2 до 1. Соответствующая программа приведена ниже.

```
k:=2;
h:=2; {шаг изменения индекса выходного массива}
for i:=1 to n do
begin     b[k]:=a[i];
          k:=k+h;
          if k>n then    begin
                              if n mod 2=0
                              then k:=n-1
                              else k:=n;
                              h:=-2
                         end
end.
```

Упражнение. Постройте программу на основе перебора элементов исходного массива, но выбрав схему перебора по одному элементу от последнего к первому.

Пример 11.28. Многочлен степени n $a_nx^n+a_{n-1}x^{n-1}+...+a_1x+a_0$ от одной переменной можно представить одномерным массивом коэффициентов этого многочлена, описанным следующим образом:

```
const nn=100; { максимальная степень многочлена равна 100 }
type polinom=array [0..nn] of real;
var       a,b:polinom; { a и b многочлены, свободные члены хранятся в
элементах a[0] и b[0], a[i] и b[i] - коэффициенты при x в i-той степени}
          n:integer; { степень многочлена }
```

Массивом a задан многочлен степени n. Массивом b задан многочлен степени m. Получить в массиве c результат умножения заданных многочленов.
Например, заданы многочлены: n=2, $3x^2+2x-1$ и m=3, $4x^3-5x^2+6x-7$. Их произведение является многочленом степени n+m=2+3=5. Его можно получить следующим образом:

$(3x^2+2x-1)(4x^3-5x^2+6x-7)=$

$=12x^5 - 15x^4 + 18x^3 - 21x^2 +$

$+ 8x^4 - 10x^3 + 12x^2 - 14x +$

$- 4x^3 + 5x^2 - 6x + 7 =$

$=12x^5 - 7x^4 + 4x^3 - 4x^2 - 20x + 7$.

Для получения произведения нужно каждый одночлен первого многочлена (обозначим его i) умножить на каждый одночлен второго многочлена (обозначим его j). В результате получится одночлен, в котором неизвестная будет в степени i+j. Останется привести подобные члены и получится результат. Подобные можно приводить по мере вычисления промежуточных одночленов. Таким образом, коэффициенты результата будут получаться по формуле $c_{i+j} = \sum_{i=0}^{n} \sum_{j=0}^{m} a_i \cdot b_j$.

Соответствующий алгоритм приведен ниже:

```
for i:=0 to n do
    for j:=0 to m do c[i+j]:=c[i+j]+a[i]*b[j].
```

Упражнение. Напишите алгоритм подстановки в многочлен на место переменной другого многочлена и раскрытия всех скобок.

Пример 11.29. Заданы два упорядоченных по возрастанию элементов одномерных массива: a размерности n и b размерности m. Требуется получить третий массив c размерности n+m, который содержал бы все элементы исходных массивов, упорядоченные по возрастанию.

Решение. Алгоритм решения этой задачи известен как «сортировка фон Неймана». Идея алгоритма состоит в следующем. Сначала анализируются первые элементы обоих массивов. Меньший элемент переписывается в новый массив. Оставшийся элемент последовательно сравнивается с элементами из другого массива. В новый массив после каждого сравнения попадает меньший элемент. Процесс продолжается до исчерпания элементов одного из массивов. Затем остаток другого массива дописывается в новый массив. Полученный новый массив упорядочен.

Для массива a будем использовать в качестве индекса переменную i, для массива b - j, для массива c - k. Описанный алгоритм может быть реализован на Паскале так:

```
k:=0; {в новом массиве пока нет элементов}
i:=1; j:=1; {сравнение начинаем с первых элементов}
while (i<=n) and (j<=m) do {пока один из массивов не закончился}
    if a[i]<b[j] {меньший элемент записываем в массив c}
    then begin k:=k+1; c[k]:=a[i]; i:=i+1 end
    else begin k:=k+1; c[k]:=b[j]; j:=j+1 end;
{если массив a не пуст, то дописываем оставшиеся элементы в массив c}
while i<=n do
begin      k:=k+1;
```

```
            c[k]:=a[i];
            i:=i+1
end;
{если массив b не пуст, то дописываем оставшиеся элементы в массив c}
while j<=m do
begin       k:=k+1;
            c[k]:=b[j];
            j:=j+1
end.
```

Часто в программировании встречается ситуация, когда требуется для выбранных элементов массива просмотреть часть этого же массива. Особенно часто эта ситуация встречается в задачах сортировки массивов. Просматриваемая часть массива называется подмассивом. Методы работы с подмассивами совпадают с методами работы с несколькими массивами.

Пример 11.30. В заданном одномерном массиве все нулевые элементы переписать в конец массива. Например, для массива a={1,0,2,3,0,4,5,0,0,0,6} должен получиться результат a={1,2,3,4,5,6,0,0,0,0,0}.

Решение. Можно воспользоваться услугами вспомогательного массива, в который вначале перепишем ненулевые элементы (вспомним пример 11.26), оставшуюся часть вспомогательного массива заполним нулями. Чтобы закончить решение, останется переписать содержимое вспомогательного массива в исходный массив. Соответствующий вариант решения приведен ниже. Здесь a - исходный массив, i - индекс массива a, b - вспомогательный массив, j - его индекс. Размерность обоих массивов - n.

```
j:=1; { указывает номер первой свободной позиции в массиве b }
{ переписываем все ненулевые элементы из b в a }
for i:=1 to n do
    if a[i]<>0 then begin b[j]:=a[i]; j:=j+1 end;
{ заполняем остаток массива b нулями }
for i:=j to n do b[i]:=0;
{ переписываем массив b в a }
a:=b.
```

Проанализируем полученное решение. Главным его недостатком является использование вспомогательного массива - нерациональное использование оперативной памяти компьютера. Попробуем обойтись без него. В данном случае можно записывать ненулевые элементы в начало исходного массива. Место записи элемента указывает переменная j, которая передвигается по массиву только после записи ненулевого элемента. При найденном нуле значение переменной j не изменяется. Переменная i будет указывать на проверяемый элемент массива. Получается, что для одного массива используются два индекса: индекс i перебирает элементы массива a как входного (исходного), а индекс j - как выходного. В этом алгоритме результат получается сразу в массиве a, не нужно тратить время на переписывание из вспомогательного алгоритма. Соответствующий вариант программы приведен ниже:

```
j:=1;
for i:=1 to n do
```

```
    if a[i]<>0 then begin a[j]:=a[i]; j:=j+1 end;
for i:=j to n do a[i]:=0.
```

Упражнения:

1. Проведите трассировку предложенных в предыдущем примере алгоритмов. Постройте для них систему тестов.

2. Постройте алгоритм для переписывания всех нулей одномерного массива в его начало. Используйте идеи рассмотренного примера.

Пример 11.31. Задан одномерный массив. Нужно преобразовать его так, чтобы все элементы, большие k, разместились в конце массива, а меньшие или равные k в начале.

Решение. Например, для массива a={16,1,5,3,8,4,7,1,3,2} и k=4 существует много перестановок элементов массива, удовлетворяющих заданным условиям. Примерами выходных данных могут быть:

1) a={1,1,2,3,3,4,5,7,8,16};
2) a={1,3,4,1,3,2,7,8,5,16};
3) a={2,1,3,3,1,4,7,8,5,16}.

Первый результат получен сортировкой массива. Методы сортировки будут рассмотрены в следующей главе. Второй результат получен с помощью вспомогательного массива, в начало которого выписывались элементы, меньшие k, при просмотре исходного массива от первых элементов к последним, а в конец - элементы, большие k. Это решение представлено следующим фрагментом программы:

```
i:=1; {индекс начальных элементов выходного массива}
j:=n; {индекс последних элементов выходного массива}
for m:=1 to n do {просматриваем все элементы исходного массива}
    if a[m]>k
    then begin b[j]:=a[m]; j:=j-1 end
    else begin b[i]:=a[m]; i:=i+1 end;
{переписываем результат в исходный массив}
a:=b.
```

Недостатком представленного решения является нерациональное использование оперативной памяти - использование вспомогательного массива. Третье решение получено без помощи вспомогательного массива. Здесь используется следующая идея. Массив начинает просматриваться с начала до обнаружения элемента, большего k, затем просмотр массива идет с конца до нахождения элемента, меньшего или равного k, найденные элементы меняются местами. После обмена просмотр возобновляется с начала массива от элемента, следующего за обмененным. Это решение приведено ниже:

```
i:=1; {индекс начальных элементов исходного массива}
j:=n; {индекс конечных элементов исходного массива}
while i<=j do
    if a[i]>k
    then    if a[j]<=k
            then    begin  r:=a[i];
                           a[i]:=a[j];
                           a[j]:=r;
```

```
                    i:=i+1;
                    j:=j-1
                end
            else j:=j-1
        else i:=i+1.
```

Для задач с несколькими возможными ответами рекомендуется выбирать такой алгоритм, который приводит к результату быстрее и не использует больших объемов дополнительной памяти. В данном случае это третий алгоритм, в котором затраты памяти сводятся к одной вспомогательной переменной r, а результат получается за один просмотр массива.

Упражнения:

1. Задан одномерный массив. Преобразуйте его так, чтобы все элементы, меньшие или равные k, разместились в конце массива, а большие k в начале.

2. Осуществите сжатие одномерного массива a длины n, удалив из него каждый i-й элемент. Конец массива после сжатия заполните нулями.

11.6.4. Решение задач четвертого класса

Поиск элемента в массиве является наиболее часто встречающимся алгоритмом. Задача заключается в поиске среди элементов одномерного массива элемента, равного заданному «аргументу поиска» a. Существует много вариантов решения этой задачи.

Вариант 1 *(линейный поиск)*. Если нет дополнительной информации о разыскиваемых данных, то остается простой последовательный просмотр массива с увеличением шаг за шагом той его части, где искомого элемента не обнаружено. Такой метод называется линейным поиском. Исходными данными для решения задачи является массив чисел. В результате получаем либо номер элемента, совпадающего с A, либо ответ: «Такого элемента нет». Решение этой задачи может закончиться по двум причинам: 1) перебрали все элементы массива и не нашли нужного элемента; 2) в процессе перебора обнаружился элемент, совпавший с A. В этом случае перебор прекращается и формируется ответ.

Первую причину окончания можно определить, проверив условие: i > n, где i - текущий элемент массива, n - количество элементов в массиве.

Вторая причина имеет два значения: «найдено» или «не найдено», поэтому ее можно изображать логическим значением, где true соответствует найдено, a false - не найдено.

Таким образом, условие окончания поиска может быть записано в виде (i>n) or f, что соответствует фразе русского языка: «Просмотрены все элементы или не найдено». Просмотр элементов массива в цикле будет выполняться в случае ложности приведенного условия. Найдем его отрицание, воспользовавшись законом де Моргана: отрицание дизъюнкции равно конъюнкции отрицаний.

not((i>n) or f) = not(i>n) and not f = (i<=n) and not f.

Программа на алгоритмическом языке Паскаль.

```
program POISK;
{ линейный поиск элемента в одномерном массиве }
const nn=12; { константа задает максимальный размер массива }
type mas1=array[1..nn] of integer; { тип массива }
```

```
var         b : mas1; { исходный массив }
            n : integer; { размер массива }
            a : integer; { число для поиска }
            i : integer; { индекс массива }
            f : Boolean; { истина, если искомый элемент найден }
begin
   write('Введите размер массива от 1 до ',nn);
   repeat     { вводим n до тех пор, }
                  readln(n);   { пока оно не попадет }
   until (1<=n) and (n<=nn); { на отрезок от 1 до nn }
   { Вводим элементы массива }
   for i:=1 to n do
   begin  write('Введите элемент массива B[',i:2,']=');
          readln(b[i])
   end;
   write('Введите искомый элемент ');
   readln(a);
   i:=1; { начинаем просмотр с первого элемента }
   f:=false; { пока не найдено }
   while (i<=n) and not f do
   { просматриваем массив, пока есть элементы и не найден искомый }
          if b[i]=a { если совпали, }
          then f:=true { то нашли, }
          else i:=i+1; { иначе перейти к следующему элементу }
   if f  { вывод результатов поиска }
   then write('Номер найденного элемента ',i)
   else write('Не нашли')
end.
```

Тестирование.

Тест 1. n=8, B[1]=2, B[2]=8, B[3]=3, B[4]=1, B[5]=9, B[6]=2, B[7]=2, B[8]=2, a=5. Результат поиска: «Не нашли».

Тест 2. n=7, B[1]=2, B[2]=8, B[3]=3, B[4]=1, B[5]=9, B[6]=8, B[7]=2, a=8. Результат поиска: «Номер найденного элемента 2».

Вариант 2 *(линейный поиск с барьером)*. В этом варианте программы применяется широко распространенный прием фиктивного элемента, или барьера, расположенного в конце массива. Использование барьера позволяет упростить условие окончания цикла, т.к. заранее ясно, что хотя бы один элемент, равный a, в массиве есть.

```
program POISK1;
{ поиск с барьером в одномерном массиве }
const nn=12; { константа задает максимальный размер массива }
type mas1=array[1..nn] of integer; { тип массива }
var         b : mas1; { исходный массив }
            n : integer; { размер массива }
            a : integer; { число для поиска }
            i : integer; { индекс массива }
```

```
begin
   write('Введите размер массива от 1 до ',nn-1);
   repeat     { вводим n до тех пор, }
          readln(n);   { пока оно не попадет }
   until (1<=n) and (n<=nn); { на отрезок от 1 до nn-1 }
   { Вводим элементы массива }
   for i:=1 to n do
   begin
          write('Введите элемент массива B[',i:2,']=');
          readln(b[i])
   end;
   write('Введите искомый элемент ');
   readln(a);
   b[n+1]:=a; {добавили барьер, равный a, в конец массива}
   i:=1; { начинаем просмотр с первого элемента }
   while b[i]<>a do i:=i+1;
   { просматриваем массив, пока не найдем }
   if i<=n  { вывод результатов поиска }
   then write('Номер найденного элемента ',i)
   else write('Не нашли')
end.
```

Тестирование.

Тест 1. n=8, B[1]=2, B[2]=8, B[3]=3, B[4]=1, B[5]=9, B[6]=2, B[7]=2, B[8]=2, B[9]=5, a=5. Результат поиска: «Не нашли».

Тест 2. n=7, B[1]=2, B[2]=8, B[3]=3, B[4]=1, B[5]=9, B[6]=8, B[7]=2, B[8]=8, a=8. Результат поиска: «Номер найденного элемента 2».

Упражнения:

1. Проведите трассировку и запомните оба варианта поиска.

2. Напишите программы поиска, в которых элементы массива перебираются от конца к началу.

Вариант 3 *(поиск методом деления пополам, или двоичный поиск)*. Поиск элемента в одномерном массиве можно значительно ускорить, если предварительно упорядочить (отсортировать) массив. Для поиска массив делится пополам и для сравнения выбирается средний элемент. Если он совпадает с искомым, то поиск заканчивается. Если средний элемент меньше искомого, то все элементы, расположенные левее его, также будут меньше искомого. Следовательно, их можно исключить из дальнейшего поиска, оставив только правую часть массива. Если средний элемент больше искомого, то отбрасывается правая часть, а остается левая. Далее повторяется процедура деления массива пополам и отбор части массива для дальнейшего поиска. Так продолжается до тех пор, пока искомый элемент не будет найден или длина части массива для поиска не станет равной нулю. Каждый раз величина той части массива, где ведется поиск, уменьшается вдвое. Поэтому максимальное число требующихся сравнений равно $]\log_2 N[$, где N - количество элементов в массиве.

Рассмотрим пример двоичного поиска.

Искомый элемент a=7. Исходный массив n=16.

Первый шаг поиска:
номер элемента: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16;
массив B={ 1, 3, 5, 6, 7,10,12,14,15,16,20,21,23,25,26,30 };
начальная граница поиска L=1, конечная граница поиска R=16;
середина части поиска i=(L+R) div 2 = 8;
поскольку B[8]=14>a=7, то отбрасываем правую часть: L=1, R=i-1=7.
Второй шаг поиска:
номер элемента: 1 2 3 4 5 6 7;
массив B={ 1, 3, 5, 6, 7,10,12 };
начальная граница поиска L=1, конечная граница поиска R=7;
середина части поиска i=(L+R) div 2 = 4;
поскольку B[4]=6<a=7, то отбрасываем левую часть: L=i+1=5, R=7.

Третий шаг поиска:
номер элемента: 5 6 7;
массив B={ 7,10,12 };
начальная граница поиска L=5, конечная граница поиска R=7;
середина части поиска i=(L+R) div 2 = 6;
поскольку B[6]=10>a=7, то отбрасываем правую часть: L=5, R=i-1=5.
Четвертый шаг поиска:
номер элемента: 5;
массив B={ 7 };
начальная граница поиска L=5, конечная граница поиска R=5;
середина части поиска i=(L+R) div 2 = 5;
поскольку B[5]=7=a=7, то искомый элемент найден в позиции 5.

Фрагмент программы на Паскале, реализующий двоичный поиск, приведен ниже:

```
l:=1; r:=n;
f:=false; {не найдено}
while (l<=r) and not f do
begin      i:=(l+r) div 2;
           if b[i]=a then f:=true {нашли}
           else if b[i]<a then l:=i+1 else r:=i-1
end.
```

Тестирование.

Тест 1: a=7, n=15, b={1,3,5,7,9,10,12,14,16,18,21,23,25,27,29}.
Ответ: искомый элемент найден в позиции 4.

Тест 2: a=6, n=15, b={1,3,5,7,9,10,12,14,16,18,21,23,25,27,29}.
Ответ: искомый элемент не найден.

11.7. Сортировка массивов

Для применения двоичного поиска массив должен быть отсортирован, т.е. элементы массива должны быть переставлены в порядке возрастания или убывания значений элементов. При решении задачи сортировки обычно выдвигается требование минимального использования дополнительной памяти, что исключает применение дополнительных массивов. Эффективность алгоритмов сортировки

оценивается числами: C - количество необходимых сравнений элементов, M - количество пересылок элементов. Они определяются некоторыми функциями от числа n сортируемых элементов. Здесь рассматриваются простые методы сортировки, которые требуют порядка n^2 сравнений. Их использование обусловлено следующими причинам:

1) простые методы хорошо подходят для разъяснения свойств большинства принципов сортировки;

2) программы, построенные на этих методах, коротки и легки для понимания. (следует помнить, что программы также занимают место в памяти);

3) хотя сложные методы требуют меньшего количество операций, эти операции более сложны, поэтому для малых n простые методы работают быстрее.

Методы сортировки можно разбить на три основных класса в зависимости от лежащего в их основе приема:

1) сортировка включением (вставкой);

2) сортировка выбором (выделением);

3) сортировка обменом («пузырьковая сортировка»).

11.7.1. Сортировка включениями

Для этой сортировки массив разделяется на две части: отсортированную и неотсортированную. Элементы из неотсортированной части поочередно выбираются и вставляются в отсортированную часть так, чтобы упорядоченность этой части не нарушилась. В начале работы отсортированная часть содержит только один элемент, а все остальные - в неотсортированной части. Таким образом, алгоритм будет состоять из n-1 прохода (n - размерность массива), каждый из которых будет включать следующие действия:

- взятие первого элемента несортированной части;

- поиск места выбранного элемента в отсортированной части: последовательное сравнение выбранного элемента с элементами отсортированной части при просмотре их с конца. Если элемент отсортированной части больше выбранного, то он пересылается на одну позицию вправо, иначе выбранный элемент записывается на свободное место в отсортированную часть.

Рассмотрим возможную реализацию этого метода.

n=6. Отсортированная часть отделена вертикальной чертой.

Исходный массив a={ 3 | 1, 9, 2, 5, 7 }.

Номера элементов: 1 2 3 4 5 6.

Начало отсортированной части - 1, конец отсортированной части - 1.

Начало неотсортированной части - 2, конец неотсортированной части - n.

Первый шаг сортировки:

выбираем первый элемент неотсортированной части: y=a[2]=1;

сравниваем его с элементами отсортированной части, двигаясь от ее конца к началу; поскольку a[1] > y=1, сдвигаем a[1] на одно место вправо;

преобразованный массив a={ | 3, 9, 2, 5, 7 };

номера элементов: 1 2 3 4 5 6;

поскольку отсортированная часть закончилась, нужно выбранный элемент записать в массив на свободное место; длина отсортированной части увеличилась на 1;

преобразованный массив a={ 1, 3 | 9, 2, 5, 7 };

номера элементов: 1 2 3 4 5 6;

начало отсортированной части - 1, конец отсортированной части - 2;

Начало неотсортированной части - 3, конец неотсортированной части - n.

Второй шаг сортировки:

выбираем первый элемент неотсортированной части: y=a[3]=9;

сравниваем его с элементами отсортированной части, двигаясь от ее конца к началу; поскольку a[2] < y=9, сравнения прекращаем и записываем выбранный элемент обратно в массив; длина отсортированной части увеличилась на 1;

преобразованный массив a={ 1, 3, 9 | 2, 5, 7 };

Номера элементов: 1 2 3 4 5 6;

Начало отсортированной части - 1, конец отсортированной части - 3;

Начало неотсортированной части - 4, конец неотсортированной части - n.

Третий шаг сортировки:

выбираем первый элемент неотсортированной части: y=a[4]=2;

сравниваем его с элементами отсортированной части, двигаясь от ее конца к началу; поскольку a[3] > y=2, сдвигаем a[3] на одно место вправо;

преобразованный массив a={ 1, 3, | 9, 5, 7 };

номера элементов: 1 2 3 4 5 6;

сравниваем a[2] > y=2; сдвигаем a[2] на одно место вправо;

преобразованный массив a={ 1, , 3 | 9, 5, 7 };

номера элементов: 1 2 3 4 5 6;

сравниваем a[1] < y=2; заканчиваем сравнения и записываем выбранный элемент на свободное место в массиве; длина отсортированной части увеличилась на 1;

преобразованный массив a={ 1, 2, 3, 9 | 5, 7 };

номера элементов: 1 2 3 4 5 6;

начало отсортированной части - 1, конец отсортированной части - 4;

начало неотсортированной части - 5, конец неотсортированной части - n.

Четвертый шаг сортировки:

выбираем первый элемент неотсортированной части: y=a[5]=5;

сравниваем его с элементами отсортированной части, двигаясь от ее конца к началу; поскольку a[4] > y=5, сдвигаем a[4] на одно место вправо;

преобразованный массив a={ 1, 2, 3, | 9, 7 };

номера элементов: 1 2 3 4 5 6;

сравниваем a[3] < y=5; заканчиваем сравнения и записываем выбранный элемент на свободное место в массиве; длина отсортированной части увеличилась на 1;

преобразованный массив a={ 1, 2, 3, 5, 9 | 7 };

номера элементов: 1 2 3 4 5 6;

начало отсортированной части - 1, конец отсортированной части - 5;

начало неотсортированной части - 6, конец неотсортированной части - n.

Пятый шаг сортировки (последний в данном примере):

выбираем первый элемент неотсортированной части: y=a[6]=7;

сравниваем его с элементами отсортированной части, двигаясь от ее конца к началу; поскольку a[5] > y=7, сдвигаем a[5] на одно место вправо;

преобразованный массив a={ 1, 2, 3, 5 | 9 };
номера элементов: 1 2 3 4 5 6;
сравниваем a[4] < y=7; заканчиваем сравнения и записываем выбранный элемент на свободное место в массиве; длина отсортированной части увеличилась на 1;
преобразованный массив a={ 1, 2, 3, 5, 7, 9 };
номера элементов: 1 2 3 4 5 6;
начало отсортированной части - 1, конец отсортированной части - n;
неотсортированной части в массиве нет.

Программа, реализующая описанный алгоритм, приведена ниже:

```
for i:=2 to n do {перебираем все элементы неотсортированной части}
begin   j:=i-1; {номер последнего элемента отсортированной части}
        y:=a[i]; {выбрали первый элемент неотсортированной части}
        f:=false; {пока не нашли для него подходящего места}
        while (j>0) and not f do { пока есть элементы в отсортированной части
и не найдено в ней место для выбранного элемента выполняем }
                if a[j]>y       then { сдвигаем a[j] на одно место вправо }
                                begin a[j+1]:=a[j]; j:=j-1 end
                                else f:=true;
                a[j+1]:=y {записываем элемент на найденное место в массив}
end.
```

Упражнения:

1. Измените алгоритм, используя для поиска места в отсортированной части просмотр от ее начала к концу.
2. Измените алгоритм, используя для поиска места в отсортированной части двоичный поиск.
3. Сравните предложенный алгоритм и алгоритмы, полученные при выполнении упражнений 1 и 2.
4. Примените любой из этих алгоритмов для сортировки части массива между элементами с индексами p и q.
5. Измените предложенный алгоритм так, чтобы сортировка шла в обратном порядке.

11.7.2. Сортировка выбором

Вначале находим в массиве элемент с минимальным значением среди элементов с индексами от 1-го до n-го и меняем найденный элемент с первым. После этого первый элемент из обработки можно исключить. На втором шаге находим минимальный элемент среди элементов с индексами от 2-го до n-го и меняем его местами со вторым элементом. Продолжаем повторять поиск минимального элемента и его обмен со всем элементами с 3-го до n-1-го.

Отсортируем с помощью этого метода массив a={ 3, 1, 9, 2, 5, 7}. Вначале найдем минимальный элемент во всем массиве и поставим его на первое место. Получим a={ 1, | 3, 9, 2, 5, 7 }. Сейчас первый элемент стоит на своем месте, поэтому его не трогаем, а отыскиваем минимальный элемент, начиная со второго до последнего. Ставим найденный элемент на второе место. Получим a={ 1, 2, | 9, 3, 5, 7 }. Сейчас два элемента стоят на своих местах, поэтому следующий шаг

поиска проводим для элементов с третьего до последнего. Получим a = { 1, 2, 3, | 9, 5, 7 }. По окончании четвертого шага получим a={ 1, 2, 3, 5, | 9, 7 }. Наконец, после пятого шага получим a = { 1, 2, 3, 5, 7, 9 }.

Таким образом, алгоритм этого метода сортировки сводится к следующему. Организуем цикл, который позволит изменять границы просматриваемой части массива. В теле этого цикла вначале находим минимальный элемент и его номер. Затем переставляем найденный элемент на первое место в просматриваемой части. Фрагмент программы, реализующей данный алгоритм, приведен ниже:

```
for i:=1 to n-1 do {i - начало обрабатываемой части массива}
begin     {поиск минимального элемента от i-го элемента до n-го}
          min:=a[i];
          imin:=i;
          for j:=i+1 to n do
                if a[j]<min then begin min:=a[j]; imin:=j end;
          {обмен местами минимального и i-го элементов}
          a[imin]:=a[i];
          a[i]:=min
end.
```

Упражнение. Постройте аналогичный алгоритм сортировки, используя поиск максимального элемента в массиве или его части.

11.7.3. Сортировка обменом

Простейшей сортировкой этого типа является «пузырьковая» сортировка. Суть этой сортировки сводится к следующему. Просматриваем по два соседних элемента, двигаясь от конца массива к началу. Если левый сосед больше правого, то меняем их местами. В результате такого просмотра минимальный элемент окажется на первом месте в массиве («всплыл» первый «пузырек»). Поскольку минимальный элемент уже стоит на своем месте, на следующем шаге просматриваем элементы с последнего до второго. После второго просмотра два элемента массива будут отсортированы. Выполнив n-1 раз просмотры пар соседних элементов, начиная с последнего до элемента с номером,равным номеру шага, получим полностью отсортированный массив.

Например, пусть a={ 3, 1, 9, 2, 5, 7 }, тогда первый просмотр даст следующие результаты:

```
3    1    9    2    5    7
                   5    7 сравниваем,
              2    5 сравниваем,
         2    9 сравниваем и меняем местами,
    1    2 сравниваем,
1    3 сравниваем и меняем местами.
```

```
Второй просмотр:  1    | 3    2    9    5    7
                                   5    7 сравниваем,
                              5    9 сравниваем и меняем
местами,
                         2    5 сравниваем,
                    2    3 сравниваем и меняем местами.
```

Третий просмотр: 1 2 | 3 5 9 7
7 9 сравниваем и меняем местами,
5 7 сравниваем,
3 5 сравниваем.

Четвертый просмотр: 1 2 3 | 5 7 9
7 9 сравниваем,
5 7 сравниваем.

Пятый просмотр: 1 2 3 5 | 7 9
7 9 сравниваем.

Завершены все n-1 (в данном случае пять) просмотров. Массив отсортирован. В данном случае четвертый и пятый просмотр оказались лишними, однако в общем случае они необходимы. Ниже приводится фрагмент программы, реализующий данный алгоритм.

```
for i:=1 to n-1 do {всплывание «пузырька» на i-ое место}
    for j:=n downto i+1 do
            if a[j-1]>a[j] then begin r:=a[j]; a[j]:=a[j-1]; a[j-1]:=r end.
```

Упражнения:

1. Перепишите фрагмент программы так, чтобы просмотр соседей осуществлялся с первого элемента массива и «тяжелые» элементы оседали в конце массива.
2. Модифицируйте предложенный вариант алгоритма сортировки обменом, чтобы убрать лишние просмотры.
3. Постройте такой тест, для которого все n-1 просмотры будут результативными, т.е. в ходе просмотра будет совершен хотя бы один обмен элементов.
4. Установите, какую задачу решает представленный ниже фрагмент программы. Сравните данный алгоритм с алгоритмом метода «пузырька».

```
for i:=1 to n-1 do
    for j:=i+1 to n do
            if a[i]>a[j] then begin r:=a[i]; a[i]:=a[j]; a[j]:=r end.
```

11.8. Двумерные массивы

Двумерный массив можно рассматривать как одномерный массив, каждый элемент которого сам является одномерным массивом. Поэтому для работы с элементами двумерного массива нужно организовать два цикла. Каждый из них отвечает за перебор значений соответствующего индекса. Для двумерного массива можно использовать те же схемы перебора, что и для одномерного, но комбинаций здесь будет в два раза больше (почему?).

Рассмотрим один из способов ввода элементов двумерного массива. Будем использовать схему перебора по одному от начала массива к концу. Соответствующий фрагмент алгоритма приведен ниже. Считаем, что массив имеет размерность $n \cdot m$.

```
for i:=1 to n do {перебираем строки двумерного массива}
    for j:=1 to m do {перебираем столбцы двумерного массива}
            read(a[i,j]).
```

В случае использования этой схемы перебора при n=3 и m=2, получаем следующий порядок ввода элементов: a[1,1], a[1,2], a[2,1], a[2,2], a[3,1], a[3,2]. Такой порядок ввода элементов называется вводом по строкам. Для ввода по столбцам поменяем местами индексы i и j в операторе ввода. Получаем ввод элементов по столбцам: a[1,1], a[2,1], a[3,1], a[2,1], a[2,2], a[2,3], a[3,1], a[3,2], a[3,3].

```
for i:=1 to n do
    for j:=1 to n do
            read(a[j,i]).
```

Упражнения:

1. Приведите другие схемы перебора, осуществляющие ввод элементов двумерного массива по столбцам.

2. Как будут вводиться элементы двумерного массива, если в программе будут использованы следующие операторы:

```
а) for i:=n downto 1 do
        for j:=m downto 1 do read(a[i,j]);
б) for i:=n downto 1 do
        for j:=m downto 1 do read(a[j,i]) ;
в) for i:=1 to n do
        for j:=m downto 1 do read(a[i,j]);
г) for i:=n downto 1 do
        for j:=1 to m do read(a[i,j]).
```

Рассмотрим задачу печати элементов двумерного массива по строкам и столбцам. Задача состоит в печати стольких строк, сколько их в двумерном массиве. Это реализуется арифметическим циклом по количеству строк в двумерном массиве. В теле этого цикла выполняются следующие работы: переход к следующей печатаемой строке и вывод элементов строки (их столько сколько столбцов). Вывод элементов строки реализуется арифметическим циклом по количеству столбцов. Соответствующий фрагмент программы приведен ниже:

```
for i:=1 to n do {перебираем строки}
begin   writeln;
        for j:=1 to m do write(a[i,j],' ') { перебираем столбцы, каждый элемент
отделяем от другого пробелом }
end.
```

Упражнения:

1. Будут ли выдавать одинаковые результаты ранее рассмотренный фрагмент вывода двумерного массива и фрагмент, приведенный ниже? Если «да», то в каких случаях? Если «нет», то почему?

```
for i:=1 to n do
begin   for j:=1 to m do write(a[i,j],' ');
        writeln
end.
```

2. Можно ли вывести двумерный массив на печать, перебирая элементы по столбцам и сохраняя вид массива?

3. Можно ли решить задачу из упражнения 2 при выводе элементов на экран? Какие дополнительные операторы, отсутствующие обычно в алгоритмических языках, нужны для решения этой задачи?

4. Как будет выведен двумерный массив на экран при использовании следующего фрагмента программы?

```
for i:=n downto 1 do
begin    writeln;
         for j:=m downto 1 do write(a[i,j],' ')
end.
```

5. Что изменится, если во внутреннем цикле предыдущего фрагмента заменить оператор вывода на следующий: write(a[j,i],' ')?

Для двумерных массивов справедливы те же классы задач, что и для одномерных, но схемы перебора организуются с помощью двух циклов. Для решения задач на двумерные массивы нужно знать некоторые соотношения для индексов. Рассмотрим двумерный массив размерностью n·n для n=6. Здесь количества строк и столбцов совпадают, поэтому такие массивы называются квадратными.

a_{11} a_{12} a_{13} a_{14} a_{15} a_{16}
a_{21} a_{22} a_{23} a_{24} a_{25} a_{26}
a_{31} a_{32} a_{33} a_{34} a_{35} a_{36}
a_{41} a_{42} a_{43} a_{44} a_{45} a_{46}
a_{51} a_{52} a_{53} a_{54} a_{55} a_{56}
a_{61} a_{62} a_{63} a_{64} a_{65} a_{66}

Если мысленно провести прямую из левого верхнего угла массива в правый нижний, то можно заметить, что все элементы, расположенные на этой линии, будут иметь совпадающие индексы: a_{11}, a_{22}, a_{33}, a_{44}, a_{55}, a_{66}. Эту линию называют *главной диагональю* двумерного массива. Если индекс строк обозначить i, а индекс столбцов - j, то в общем случае факт принадлежности элемента главной диагонали можно записать так: i=j, или так: i-j=0. Для индексов элементов, расположенных ниже главной диагонали, справедливо соотношение i>j. Для индексов элементов, расположенных выше главной диагонали, справедливо соотношение i<j.

Индексы элементов, расположенных на линиях, параллельных главной диагонали, удовлетворяют соотношениям |i-j|=k, где k - номер линии по отношению к главной диагонали. Например, для индексов элементов a_{31} a_{42} a_{53} a_{64}, расположенных на 2 линии ниже главной диагонали, справедливо i-j=2, а для индексов элементов a_{13} a_{24} a_{35} a_{46}, расположенных на две линии выше главной диагонали, справедливо i-j=-2.

Линию, соединяющую правый верхний угол массива с левым нижним, называют *побочной диагональю*. Для индексов элементов побочной диагонали: a_{16} a_{25} a_{34} a_{43} a_{52} a_{61}, справедливо соотношение i+j=n+1, где n - количество строк и столбцов массива (в примере n=6). Для индексов элементов, расположенных выше побочной диагонали, справедливо i+j<n+1, а для индексов элементов, расположенных ниже побочной диагонали, справедливо i+j>n+1.

Индексы элементов, расположенных на линиях параллельных побочной диагонали, удовлетворяют соотношению 2<=i+j<=2·n. Например, для индексов элементов: a_{14} a_{23} a_{32} a_{41}, расположенных на 2 линии выше побочной диагонали,

справедливо соотношение i+j=5, а для индексов элементов a_{36} a_{45} a_{54} a_{63}, расположенных на 2 линии ниже побочной диагонали, справедливо соотношение: i+j=9.

Пример 11.32. Сформировать двумерный массив n·n указанного вида для произвольного n. Для n=4 формируемый массив имеет вид

1 0 0 0
2 1 0 0
3 2 1 0
4 3 2 1.

Решение. Заметим, что каждая строка начинается с элемента, значение которого совпадает с номером строки. Затем оно уменьшается с ростом номера столбца. Все элементы, лежащие выше главной диагонали, равны нулю. Отсюда легко догадаться, что в теле циклов перебора повторяется оператор a[i,j]:=i-j+1. Для перебора строк и столбцов используем вложенные циклы.

```
for i:=1 to n do
   for j:=1 to n do
         if i<=j
         then a[i,j]:=i-j+1
         else a[i,j]:=0.
```

Упражнение. Воспользовавшись формулой n-i+1, получите на основе предыдущего решения массив вида

4 0 0 0
3 4 0 0
2 3 4 0
1 2 3 4.

Пример 11.33. Сформировать двумерный массив n·n указанного вида для произвольного n. Для n=4 формируемый массив имеет вид

0 0 0 4
0 0 4 3
0 4 3 2
4 3 2 1.

Решение. Можно заметить, что все элементы, расположенные выше побочной диагонали, равны нулю, а расположенные ниже связаны с индексами соотношением 2·n-(i+j)+1. Соответствующий фрагмент программы имеет вид

```
for i:=1 to n do
   for j:=1 to n do
         if i+j>=n+1
         then a[i,j]:=2*n-(i+j)+1
         else a[i,j]:=0.
```

Упражнения:

1. Постройте фрагменты программ для формирования массивов:

1	2	3	4	1	2	3	4	1	2	3	4
0	1	2	3	2	3	4	0	2	3	4	1
0	0	1	2	3	4	0	0	3	4	1	2
0	0	0	1	4	0	0	0	4	1	2	3.

2. Придумайте аналогичные виды массивов и постройте для них программы. Попробуйте сформулировать алгоритм решения таких задач.

Пример 11.34. Сформировать двумерный массив n·n указанного вида для произвольного n. Для n=4 формируемый массив имеет вид

0 1 0 2
3 0 4 0
0 5 0 6
7 0 8 0.

Решение. Легко заметить, что ненулевое значение имеют элементы, сумма индексов которых нечетна. Для формирования значения можно использовать дополнительную переменную. Для перебора элементов будем использовать два вложенных цикла. Применим схему перебора от последних элементов к первым. Она не является необходимой для этой задачи, а применена здесь в качестве примера.

```
k:=2*n; {значение последнего элемента}
for i:=n downto 1 do
   for j:=n downto 1 do
         if (i+j) mod 2 =1
         then begin a[i,j]:=k; k:=k-1 end
         else a[i,j]:=0.
```

Упражнения:

1. Перепишите фрагмент, применив схему перебора от первых элементов к последним.

2. Установите, какую задачу решает фрагмент, приведенный ниже.

```
k:=2*n;
for i:=n downto 1 do
   for j:=1 to n do
         if (i+j) mod 2 =1
         then begin a[i,j]:=k; k:=k-1 end
         else a[i,j]:=0.
```

3. Напишите фрагмент программы, печатающий шахматную доску.

Пример 11.35. Сформировать двумерный массив n·n указанного вида для произвольного n. Для n=5 формируемый массив имеет вид

1 0 0 0 1
1 1 0 1 1
1 1 1 1 1
1 1 1 1 1
1 1 1 1 1.

Здесь нули располагаются одновременно и над главной, и над побочной диагоналями. Соединив соответствующие условия конъюнкцией и выбрав схему перебора от первых к последним элементам, получаем фрагмент программы:

```
for i:=1 to n do
   for j:=1 to n do
         if (i<j) and (i+j<n+1)
         then a[i,j]:=0
         else a[i,j]:=1.
```

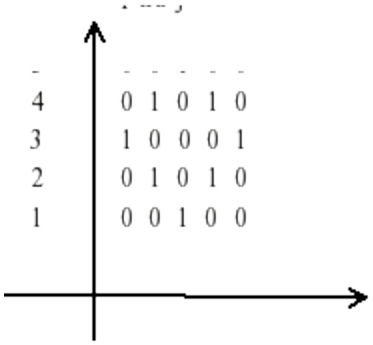
Пример 11.36. Сформировать двумерный массив n·n указанного вида для произвольного нечетного n. Для n=5 формируемый массив имеет вид

0 0 1 0 0
0 1 0 1 0
1 0 0 0 1
0 1 0 1 0
0 0 1 0 0.

Решение 1. Можно решить эту задачу, используя описанные ранее идеи. Заметим, что единицы в массиве-результате составляют ромб, две стороны которого параллельны главной диагонали и отстоят от нее на (n+1) div 2-1. Обе эти диагонали можно описать уравнением |i-j|=(n+1) div 2-1. Две другие стороны ромба параллельны побочной диагонали массива. Диагональ, лежащая выше побочной диагонали, описывается уравнением | i+j| =n-(n+1) div 2+2, а диагональ, лежащая ниже побочной, - | i+j| =n+(n+1) div 2. Это решение можно оформить следующим фрагментом:

```
k:=(n+1) div 2;
for i:=1 to n do
   for j:=1 to n do
         if (abs(i-j)=k-1) or (abs(i+j)=n-k+2) or (abs(i+j)=n+k)
         then a[i,j]:=1
         else a[i,j]:=0.
```

Решение 2. Можно подойти к решению этой задачи по-другому. Представим, что единицы в массиве изображают график функции | x| +| y| =k-1, смещенный относительно начала координат на k (рис. 11.2). Обозначим ось x через i, а ось y через j. Тогда все элементы массива, равные единице, будут удовлетворять уравнению | i-k| +| j-k| =k-1.



Ниже приводится фрагмент, реализующий решение 2:

```
for i:=1 to n do
   for j:=1 to n do
         if abs(i-k)+abs(j-k)=k-1
         then a[i,j]:=1
         else a[i,j]:=0.
```

Пример 11.37. Найти индексы максимального элемента двумерного массива.

Решение. Идея решения этой задачи совпадает с идеей решения задачи для одномерного массива. Отличие заключается в необходимости для двумерного массива вложенных циклов перебора. Фрагмент программы приведен ниже:

```
imax:=1; jmax:=1; {предполагаем максимальным первый элемент}
for i:=1 to n do
   for j:=1 to n do
         if a[imax,jmax]<a[i, j]
         then begin imax:=i; jmax:=j end.
```

Пример 11.38. Переставить строки и столбцы двумерного массива так, чтобы его максимальный элемент оказался в левом верхнем углу.

Решение сводится:

1) к поиску индексов максимального элемента (пример 11.36);

2) к обмену местами первой строки и строки imax;

3) к обмену местами столбца 1 и столбца jmax. Шаги 2 и 3 можно поменять местами без изменения результата.

Фрагмент программы приведен ниже:

```
{поиск индексов максимального элемента}
imax:=1; jmax:=1; {предполагаем максимальным первый элемент}
for i:=1 to n do
   for j:=1 to n do
         if a[imax,jmax]<a[i, j]
         then begin imax:=i; jmax:=j end.
{обмен местами первой и imax строк}
for i:=1 to n do
begin r:=a[1,i]; a[1,i]:=a[imax,i]; a[imax,i]:=r end;
{обмен местами первого и jmax столбцов}
for i:=1 to n do
begin r:=a[i,1]; a[i,1]:=a[i,jmax]; a[i,jmax]:=r end.
```

Пример 11.39. Транспонировать двумерный массив, т.е. переставить местами его строки и столбцы. Например, для исходного массива:

```
1 2 3 получить:     1 4 7
4 5 6               2 5 8
7 8 9               3 6 9.
```

Решение. Из приведенного примера хорошо видно, что диагональные элементы в результате обмена остаются на своих местах, обмениваются местами элементы, расположенные симметрично относительно главной диагонали.

```
for i:=1 to n do {перебираем все строки массива}
   for j:=1 to i-1 do {перебираем элементы до главной диагонали}
   begin r:=a[i,j]; a[i,j]:=a[j,i]; a[j,i]:=r end.
```

Пример 11.40. Транспонировать двумерный массив, перебирая элементы параллельно главной диагонали.

Решение. Для таких диагоналей справедливо равенство $|i-j|=k$. Нужно перебрать элементы, расположенные ниже главной диагонали, обменяв их с симметричными относительно главной диагонали элементами. Следовательно, k будет изменяться от 1 до n-1. Номер строки для элементов, расположенных на

диагонали, параллельной главной, изменяется от k+1 до n, а номер столбца - от 1 до n-k. В теле цикла меняем местами симметричные элементы.

```
for k:=1 to n-1 do
   for i:=k+1 to n do
   begin r:=a[i,i-k]; a[i,i-k]:=a[i-k,i]; a[i-k,i]:=r end.
```

Упражнение. Установите, какие задачи решают приведенные фрагменты:

а)
```
for k:=1 to n-1 do
        for j:=1 to n-k do
        begin r:=a[k+j,j]; a[k+j,j]:=a[j,k+j]; a[j,k+j]:=r end;
```

б)
```
for i:=n-i+j to n do
        for j:=1 to n-i+j do
        begin r:=a[i,j]; a[i,j]:=a[j,i]; a[j,i]:=r end.
```

Пример 11.41. Задан двумерный массив n·n. Поменять местами строку с номером k и столбец с номером p этого массива.

Решение 1. Задача просто решается при k=p. Например, при n=3, k=p=2 и исходном массиве:

1 2 3
4 5 6
7 8 9.

Для получения ответа меняем элементы 2 и 4, 8 и 6, 5 остается на своем месте. Получаем ответ:

1 4 3
2 5 6
7 6 9.

Однако при k<>p возникают трудности с элементом, стоящим на пересечении строки k и столбца p. Если применить тот же алгоритм для n=3, k=2, p=3 к тому же исходному массиву, то меняются местами элементы 4 и 3, 5 и 6, 5 и 9, что приводит к неверному ответу: в строке появился элемент 9, которого не было в исходной строке:

1 2 4
3 6 9
7 8 9.

Для преодоления этой проблемы можно применить такой прием:

1) переставить строку k на место строки p:

1 2 3
7 8 9
4 5 6.

2) поменять местами строку p и столбец p. Сейчас при таком обмене трудностей не возникнет:

1 2 4
7 8 5
3 9 6.

3) переставим строки k и p, получим ответ:

1 2 4
3 9 6
7 8 5.

Соответствующий фрагмент на Паскале приводится ниже:

```
{обмен местами строк с номерами k и p}
for i:=1 to n do
begin r:=a[k,i]; a[k,i]:=a[p,i]; a[p,i]:=r end;
{обмен местами строки и столбца с номером p}
for i:=1 to n do
begin r:=a[p,i]; a[p,i]:=a[i,p]; a[i,p]:=r end;
{обмен местами строк с номерами k и p}
for i:=1 to n do
begin r:=a[k,i]; a[k,i]:=a[p,i]; a[p,i]:=r end;
```

Решение 2. Недостатком предыдущего решения является большое количество циклов. Попробуем решить задачу по-другому, сократив количество используемых циклов. В задаче используется понятие «переставляемые элементы». Их несколько в строке k и столбце p, поэтому переставляемые элементы строки k обозначим переменной i, а столбца p - j. Однако элемент, стоящий на пересечении строки k и столбца p, переставляться не должен. Поэтому окончательно имеем: переставляемые элементы строки k - 1<=i<=n, кроме i=p; переставляемые элементы столбца p - 1<=j<=n, кроме j=k. В соответствии с этим можно организовать цикл перебора, пока есть переставляемые элементы. В теле цикла элементы меняются местами или пропускаются, если они не являются переставляемыми. Фрагмент решения, реализующий описанную идею, приведен ниже.

```
i:=1; j:=1;
while (i<=n) and (j<=n) do
   if (i<>p) and (j<>k)
   then   begin  r:=a[j,p]; a[j,p]:=a[k,i]; a[k,i]:=r;
                 i:=i+1; j:=j+1
          end
   else   begin  if i=p then i:=i+1;
                 if j=k then j:=j+1
          end.
```

Упражнение. Выполните трассировку последней программы.

Пример 11.42. Подсчитать, в скольких строках двумерного массива n·n, содержащего целые числа, встречается число 2.

Решение. Из условия задачи следует, что требуется перебрать все строки массива (цикл по строкам). Для каждой строки нужно проверить, есть ли в ней число 2. Эта часть решения реализуется поисковым циклом. Вкладывая поисковый цикл в цикл перебора строк, получаем решение задачи. Ниже приводится фрагмент программы на Паскале:

```
s:=0; {количество элементов равных 2}
for i:=1 to n do {цикл по строкам}
begin      j:=1; f:=false;
           while (j<=n) and not f do {поисковый цикл}
                 if a[i,j]=2 then f:=true else j:=j+1;
```

```
            if f then s:=s+1 {проверка результатов поиска}
    end.
```

Пример 11.43. Найти суммы элементов строк для заданного двумерного массива n·n.

Решение. В этой задаче исходные данные представлены в виде двумерного массива. Результат также является массивом, но одномерным. Для элементов одномерного массива не нужно в данной задаче заводить отдельного индекса, так как элементы получаются синхронно с обработкой строк исходного массива. Фрагмент программы на Паскале может быть таким:

```
    for i:=1 to n do {перебираем строки}
    begin   b[i]:=0; {начальное значение суммы i-ой строки}
            for j:=1 to n do b[i]:=b[i]+a[i,j]
    end.
```

Пример 11.44. Положительные суммы элементов строк двумерного массива n·n записать в массив b.

Решение. Эта задача отличается от предыдущей тем, что не все суммы строк двумерного массива записываются в ответ, поэтому для работы с массивом b потребуется дополнительный индекс. Фрагмент программы приведен ниже:

```
    k:=0; {пока нет положительных сумм элементов строк}
    for i:=1 to n do {перебор строк двумерного массива}
    begin   s:=0; {начальное значение суммы i-ой строки}
            for j:=1 to n do s:=s+a[i,j];
            if s>0 then begin k:=k+1; b[k]:=s end
    end.
```

Упражнение. Перепишите фрагмент предыдущей программы так, чтобы не использовать дополнительную переменную s.

Пример 11.45. Найти максимальную сумму среди сумм тех столбцов целочисленного двумерного массива n·n, диагональный элемент которых положительный.

Решение. Перебираем столбцы (i). Для каждого столбца (j), проверяем положителен ли его диагональный элемент (i=j). Если диагональный элемент положителен, то находим сумму элементов этого столбца и проверяем ее на максимум. При поиске максимальной суммы возникает проблема присваивания начального значения, которое заранее неизвестно, так как заранее неизвестен столбец с положительным элементом на диагонали. Поэтому приходится заводить указатель, указывающий, в первый ли раз вычислено нужное значение.

```
    p:=0; {нет начального значения для поиска максимума}
    for i:=1 to n do {перебор столбцов}
    if a[i,i]>0
    then    begin  s:=0;
                   for j:=1 to n do s:=s+a[j,i];
                   if p=0
                   then begin max:=s; p:=1 end { установлено начальное значение
для максимума }
                   else if max<s then max:=s
            end.
```

Пример 11.46. Отсортировать элементы двумерного массива по элементам второй строки.

Решение. Исходный массив Результат

1 2 3 4 5	5 2 4 3 1
9 3 7 3 1	1 3 3 7 9
6 7 8 9 1	1 7 9 8 6.

От сортировки одномерного массива этот случай отличается только тем, что переставлять нужно не два сравниваемых элемента, а два столбца:

```
for i:=1 to n-1 do
for j:=i+1 to n do
   if a[2,i]>a[2,j]
   then   for k:=1 to n do
          begin r:=a[k,i]; a[k,i]:=a[k,j]; a[k,j]:=r
          end.
```

Пример 11.47. Найти произведение матрицы n·k на матрицу k·m.

Решение. Матрицы представляем в виде двумерных массивов. В результате получается матрица - двумерный массив n·m, где элемент $c_{ij} = \sum_{p=1}^{k} a_{ip} \cdot b_{pj}$, т.е. каждый элемент ответа равен сумме произведений элементов i-й строки массива a на соответствующие элементы j-го столбца массива b.

```
for i:=1 to n do
for j:=1 to m do
begin     c[i,j]:=0;
          for p:=1 to k do c[i,j]:=c[i,j]+a[i,k]*b[k,j]
end.
```

11.9. Решение задач с использованием массивов

В этом разделе приводятся примеры решения более сложных задач с использованием массивов. Для их решения требуется хорошо знать и умело применять идеи, рассмотренные в предыдущих разделах.

Пример 11.48. От острова Буяна до царства славного Салтана месяц пути. Капитан ежедневно записывает в вахтенный журнал пройденное расстояние. Определить, в какую десятидневку был пройден больший путь.

Решение 1. Вахтенный журнал можно представить в виде одномерного массива, содержащего 30 элементов. Отсюда получается решение.

```
s1=0;
for i:=1 to 10 do s1:=s1+a[i];
s2:=0;
for i:=11 to 20 do s2:=s2+a[i];
s3:=0;
for i:=21 to 30 do s3:=s3+a[i];
write('наибольший путь был пройден в ');
if s1>s2
then      if s1>s3
          then write('первую')
```

```
            else write('третью')
    else    if s2>s3
            then write('вторую')
            else write('третью');
    write(' десятидневку ').
```

Решение 2. Можно не дожидаться получения пройденных расстояний по всем десятидневкам, а сразу определять максимальное. Для этого придется завести специальную переменную, чтобы хранить номер десятидневки, претендующей на максимальное пройденное расстояние.

```
d=0; {номер десятидневки}
p:=0; {максимальное пройденное расстояние}
s:=0; {расстояние, пройденное в текущую десятидневку}
for i:=1 to 10 do s:=s+a[i];
if p<s then begin p:=s; d:=1 end;

s:=0;
for i:=11 to 20 do s:=s+a[i];
if p<s then begin p:=s; d:=2 end;

s:=0;
for i:=21 to 30 do s:=s+a[i];
if p<s then begin p:=s; d:=3 end;
```

В приведенном решении видны повторяющиеся части. Организуем для их выполнения цикл.

Решение 3.

```
n:=1; m:=10; {начало и конец текущей десятидневки}
d=0; {номер десятидневки}
p:=0; {максимальное пройденное расстояние}
for i:=1 to 3 do
begin       s:=0;
            for j:=n to m do s:=s+a[j]; {нашли расстояние за десятидневку}
            n:=m+1; {нашли начало следующей десятидневки}
            m:=m+10; {нашли конец следующей десятидневки}
            if p<s then begin p:=s; d:=i end {нашли максимальное
расстояние}
end.
```

Можно отказаться от дополнительных переменных m и n, если заметить, что номер дня вычисляется по формуле $10\cdot(i-1)+j$, где i - номер десятидневки, а j - номер дня в текущей десятидневке.

Решение 4.

```
d=0; {номер десятидневки}
p:=0; {максимальное пройденное расстояние}
for i:=1 to 3 do
begin       s:=0;
            for j:=1 to 10 do s:=s+a[10*(i-1)+j];
```

```
            if p<s then begin p:=s; d:=i end
    end.
```

Заметим, что индексное выражение при обращении к элементу одномерного массива в предыдущем решении аналогично выражению при обращении к двумерному, поэтому можно представить вахтенный журнал в виде двумерного массива, где строки будут изображать десятидневки, а столбцы - номера дней в текущей десятидневке. Получаем следующее решение:

Решение 5.

```
d=0; {номер десятидневки}
p:=0; {максимальное пройденное расстояние}
for i:=1 to 3 do
begin       s:=0;
            for j:=1 to 10 do s:=s+a[i,j];
            if p<s then begin p:=s; d:=i end
end.
```

Пример 11.49. Заданы два одномерных массива. Нужно получить третий массив, в который переписать те элементы из первых двух, которые имеются как в первом, так и во втором массиве, при условии, что элементы как в первом массиве, так и во втором массиве не повторяются.

Решение. Пусть массив a содержит n элементов, для обращения к которым будем использовать индекс i, массив b содержит m элементов, для обращения к которым будем использовать индекс j. В этом случае в массиве c может оказаться min(n,m) элементов. Для обращения к элементам массива c будем использовать индекс k. Для решения используем следующую идею. Просмотрим массив a. Для каждого выбранного элемента попытаемся отыскать совпадающий с ним элемент массива b. Если такой элемент в массиве b будет найден, то запишем его в массив c.

```
k:=0; {пока массив c пуст}
for i:=1 to n do {просматриваем массив a поэлементно}
begin       j:=1; f:=false; {начальные значения для поиска в массиве b}
while (j<=m) and not f do
                    if a[i]=b[j]
                    then begin k:=k+1; c[k]:=b[j]; f:=true end
                    else j:=j+1
end.
```

Упражнения:

1. Какой массив c получится, если в предыдущем фрагменте заменить логическое выражение в условном операторе на a[i]<>b[j]?

2. Решает ли приведенный ниже фрагмент задачу примера 11.48? В чем его недостатки и достоинства?

```
k:=0;
for i:=1 to n do
begin       p:=0;
            for j:=1 to m do
                    if a[i]=b[j] then p:=1;
            if p=1 then begin k:=k+1; c[k]:=a[i] end;
```

```
end.
```

3. Почему размерность массива c должена быть размерности не меньше min(n,m)?

Пример 11.50. Заданы два одномерных массива. Нужно получить третий массив, из тех элементов первых двух, которые имеются хотя бы по разу в первом или во втором массиве, при условии, что элементы как в первом массиве, так и во втором массиве не повторяются.

Решение. Пусть массив a содержит n элементов, для обращения к которым будем использовать индекс i, массив b содержит m элементов, для обращения к которым будем использовать индекс j. В этом случае в массиве c может оказаться n+m элементов. Для обращения к элементам массива c будем использовать индекс k. Поскольку по условию задачи все элементы, которые встретились в массиве a, должны быть в ответе, то вначале перепишем в массив c все n элементов массива a. Затем просматриваем все элементы массива b и проверяем не имеются ли они среди первых n элементов массива c, куда на первом шаге были переписаны элементы массива a. Если проверяемый элемент массива b отсутствует в массиве c, то он заносится на последнее место в массив c.

```
for i:=1 to n do c[i]:=a[i]; {переписали массив a в массив c}
k:=n; {в массиве c уже имеется n элементов}
for j:=1 to m do
begin     i:=1; f:=false;
          while (i<=n) and not f do
                    if b[j]=c[i] then f:=true else i:=i+1;
          if not f then begin {1} k:=k+1; c[k]:=b[j] end
   {если не найден, то записываем в массив c}
end.
```

Упражнения:

1. Как будет работать программа, если операторы отмеченные комментарием {1} подставить вместо оператора f:=true, а условный оператор за телом цикла отбросить?

2. В чем выгода изменения индекса k до присваивания значения элементу массива c? Можно ли изменять этот индекс после присваивания значения элементу c[k]?

3. Что получится в результате, если в теле поискового цикла условие b[j]=c[i] заменить на условие b[j]<>c[i]?

4. В массив c перепишите те элементы, которые есть в массиве a, но отсутствуют в массиве b. Будем считать, что в самих массивах повторяющиеся элементы отсутствуют.

Пример 11.51. Задан одномерный целочисленный массив. Найти количество различных элементов в нем. Например, в массиве {1, 2, 1, 3, 2, 1} это количество равно 3.

Решение. Для каждого элемента от 2-го до n-го (i=2,3,...,n) проверяется, нет ли ему равных среди элементов, стоящих левее (j=1,2,...,i-1). Если таковых нет, то счетчик количества различных элементов увеличивается на 1.

```
c:=0; {количество различных элементов в массиве}
for i:=2 to n do
```

```
begin
   j:=1; f:=false;
   while (j<=i-1) and not f do
         if a[i]=a[j] then f:=true else j:=j+1;
   if not f then c:=c+1.
```

Пример 11.52. Задан одномерный целочисленный массив. Найти количество элементов, которые входят в него только один раз. Например, в массиве {1, 2, 1, 3, 2, 1} это количество равно 1.

Решение. Для каждого элемента массива просматриваем все оставшиеся. Если среди оставшихся элементов нет равных проверяемому, то счетчик количества входящих в массив только один раз элементов увеличивается на 1.

```
d:=0; {количество элементов, входящих в массив только один раз}
for i:=1 to n do
begin
   j:=1; f:=false;
   while (j<=n) and not f do
         if i<>j {проверяемый элемент пропускаем}
         then if a[i]=a[j] then f:=true else j:=j+1
         else j:=j+1;
   if not f then d:=d+1.
```

Пример 11.53. Задан одномерный целочисленный массив. Найти количество элементов, которые входят в него более чем один раз. Например, в массиве {1, 2, 1, 3, 2, 1} это количество равно 2.

Решение. Для каждого элемента массива i, если ему нет равных слева (j=1, ..., i-1), разыскиваются равные справа (j=i+1, ..., n). Если справа обнаруживается элемент, равный просматриваемому, то счетчик количества таких элементов увеличивается на 1.

Решение этой задачи можно получить на основе решений предыдущих задач. Для этого нужно из результата решения задачи 11.51 вычесть результат решения задачи 11.52.

Упражнения:

1. Запишите на Паскале первый вариант решения задачи 11.53.
2. Проверьте справедливость второго варианта решения задачи 11.53.

Пример 11.54. В одномерном массиве хранятся результаты социологического опроса. Написать программу для подсчета количества результатов, отклоняющихся от среднего не более чем на 7%.

Решение. Решим задачу, используя два цикла. В первом находится среднее арифметическое значение. Во втором подсчитывается количество элементов, попавших в интервал [0.93·ср_ар.; 1.07·ср_ар.]. Сумму элементов массива можно найти в момент ввода значений.

```
program task11_54;
var      a : array [1..100] of integer; { массив для хранения результатов
социологического опроса }
         n : integer; { количество результатов опроса }
         as : real; { среднее значение }
         i : integer; { индекс элементов массива }
```

```
            m : integer; { количество результатов, отклоняющихся от среднего не
более чем на 7% }
    begin
        write('Введите количество результатов опроса ');
        readln(n);
        write('Введите ',n,' результатов опроса ');
        as := 0;
        for i:=1 to n do
        begin  read(a[i]);
               as := as+a[i]
        end;
        as := as/n; { Найдено среднее арифметическое значение }
        m := 0;
        for i:=1 to n do
               if (0.93*as<=a[i]) and (a[i]<=1.07*as)
               then m := m+1;
        write('результат равен ',m)
    end.
```

Пример 11.55. В массиве хранятся данные о расходе электроэнергии в больнице. Найти:

- среднемесячный расход;
- минимальный и максимальный расход;
- количество дней с расходом, превышающим средний.

Решение. Решение получается как комбинация известных решений.

```
    program task11_55;
    var         a : array [1..30] of real; { расход энергии за месяц }
                i,j : integer; { индексы массива }
                as : real; { среднемесячный расход }
                max,min : real; { max и min элементы массива }
                n : integer; { количество дней с расходом, превышающим средний }
    begin
        write(' Введите ежедневные данные о расходе ',
               'электроэнергии за месяц ');
        for i:=1 to 30 do read(a[i]);
        { среднемесячный расход }
        as := 0;
        for i:=1 to 30 do as := as+a[i];
        as := as/30;
        { max и min расход }
        max := a[1]; min := a[1];
        for i:=2 to 30 do
        begin  if max < a[i] then max := a[i];
               if min > a[i] then min := a[i]
        end;
        { количество дней с расходом, превышающим средний}
        n := 0;
```

```
    for i:=1 to 30 do
    if a[i]>as then n := n+1;
    writeln('результаты: ',as,max,min,n)
end.
```

Пример 11.56. Чтобы выявить самого драчливого разбойника, Али-Баба провел турнир, во время которого каждый его разбойник дрался с каждым из оставшихся. За победу присваивалось 7 очков, за ничью - 5, за поражение - 2. Результаты турнира сведены в таблицу. Написать программу, определяющую победителя турнира.

Решение. Результаты турнира удобно представить в виде двумерного массива. По диагонали этого массива запишем нули, так как сам с собой разбойник не дерется. Если Али выиграл у Гасана, то в строке Али на пересечении со столбцом Гасан появится 7, а в строке Гасан на пересечении со столбцом Али появится 2. Нетрудно заметить, что клетки, содержащие результаты одного поединка, расположены в клетках, симметричных относительно главной диагонали. При таком представлении результатов задача сводится к нахождению сумм элементов строк, а затем поиску максимальной суммы.

	Али	Гасан	Аладин	Синдбад	Сумма набранных очков
Али	0	7	2	5	14
Гасан	2	0	2	7	11
Аладин	7	7	0	5	19
Синдбад	5	2	5	0	12

Решение на Паскале запишется так:

```
program task11_56;
const m=10; { максимально возможное количество участников турнира }
var     n : integer; { реальное количество участников турнира }
        a : array [1..m,1..m] of integer; { турнирная таблица }
        b : array [1..m] of integer;  { суммы элементов строк }
        i,j : integer; { индексы элементов массива }
        k : integer; {индекс участника, набравшего наибольшее количество
очков}
begin
    write('Введите n - количество участников турнира ');
    readln(n);
    write('Введите результаты турнира ');
    for i:=1 to n do
    for j:=1 to n do read(a[i,j]);
    { Находим суммы строк и записываем их в массив b }
    for i:=1 to n do
    begin  b[i] := 0;
```

```
            for j:=1 to n do b[i] := b[i]+a[i,j];
    end;
    k := 1;
    for i:=2 to n do
            if b[k]<b[i]
            then k := i;
    write('Победил разбойник с номером ',k);
end.
```

Пример 11.57. В одномерном массиве с четным количеством элементов (2·n) находятся координаты n точек плоскости. Они располагаются в следующем порядке: x1, y1, x2, y2, x3, y3 и т.д. Определить минимальный радиус окружности с центром в начале координат, которая содержит все точки, и номера наиболее удаленных друг от друга точек.

Решение. Для определения минимального радиуса окружности найти точку, наиболее удаленную от начала координат. Для решения второй части задачи надо последовательно перебирать все пары точек и отобрать пару точек, наиболее удаленных друг от друга. Решение задачи осложняется тем, что шаг изменения индекса равен 2. Решение на Паскале приводится ниже:

```
program task11_57;
var     n : integer; { количество точек }
        a : array [1..100] of real; { координаты точек }
        i,j : integer; { индексы массива }
        k : real; { кандидат на максимум }
        k1 : real; { очередное расстояние }
        i1,j1 : integer; { номера наиболее удаленных точек }
begin
    writeln('Введите n - количество точек ');
    readln(n);
    write('Введите ',2*n,' чисел - координат точек ');
    i := 1;
    while i<2*n do
    begin  read(a[i],a[i+1]);
            i := i+2
    end;
    { нахождение радиуса окружности }
    k := sqrt(sqr(a[1]-0)+sqr(a[2]-0));
    i:= 3;
    while i<2*n do
    begin  k1 := sqrt(sqr(a[i]-0)+sqr(a[i+1]-0));
            if k<k1
            then k := k1;
            i := i+2
    end;
    write('Минимальный радиус окружности равен ',k);
    k := 0; { Наибольшее расстояние между точками равно нулю }
    i := 1; i1 := 0; j1 := 0;
```

```
    while i<2*n-2 do
    begin  j := i+2;
           while j<2*n do
           begin  k1 := sqrt(sqr(a[i]-a[j])+sqr(a[i+1]-a[j+1]);
                  if k<k1
                  then begin k := k1; i1 := i; j1 := j end;
                  j := j+2
           end
           i := i+2
    end;
    write('Наиболее удалены точки с номерами: ',i1,' и ',
              j1,'. С координатами ',i1,':',a[i1],' ',a[i1+1],
              j1,':',a[j1],' ',a[j1+1])
end.
```

Задача 11.58. В записной книжке Незнайки записаны в алфавитном порядке дни рождения всех жителей Цветочного города. Написать программу, располагающую эти даты в хронологическом порядке.

Решение. Для представления записной книжки Незнайки будем использовать два массива. В первом в алфавитном порядке хранятся имена жителей Цветочного города, во втором - в элементах с соответствующими индексами записаны даты рождения жителей в виде: 4 цифры года, 2 цифры месяца, 2 цифры дня. Например:

Массив 1	Массив 2
Авоська	19901011
Винтик	19900512
Знайка	19900506

Содержимое массивов в примере обозначает, что Авоська родился 11 октября 1990 года, Винтик - 12 мая 1990 года, Знайка - 6 мая 1990 года. Для решения задачи необходимо второй массив рассортировать по возрастанию. При этом местами нужно менять элементы не только второго массива, но и первого. После такой сортировки получим:

Массив 1	Массив 2
Знайка	19900506
Винтик	19900512
Авоська	19901011

Составлем программу:

```
program task11_58;
var     n : integer; { количество жителей Цветочного города }
        a : array [1..100] of string; { массив имен }
        b : array [1..100] of longint; { массив дат }
    { тип longint содержит восьмизначные целые числа. Разрешен в Turbo
Pascal. Здесь используется для представления дат. }
        i,j : integer; { индексы }
        r : string; { рабочая переменная }
        r1 : longint; { рабочая переменная }
```

```
begin
   write('Введите количество жителей Цветочного города ');
   readln(n);
   for i:=1 to n do
   begin  write('Введите имя и дату рождения жителя ',
                'в формате: имя ггггммдд ');
          read(a[i],b[i])
   end;
   for i:=1 to n-1 do
   for j:=i+1 to n do
          if b[i]>b[j]
          then    begin  r := a[i]; a[i] := a[j]; a[j] := r;
                         r1 := b[i]; b[i] := b[j]; b[j] := r1;
                  end;
   writeln('Результаты работы: ');
   for i:=1 to n do
          writeln(a[i],' ',b[i]);
end.
```

Пример 11.59. В одномерный массив Незнайка записал цвет глаз всех жителей Цветочного города. Написать программу, определяющую, какой цвет глаз у жителей города встречается чаще всего.

Решение. Для решения задачи просматриваем исходный массив. Обнаруженный цвет глаз выписываем во вспомогательный массив, если его там нет, или прибавляем единицу к соответствующему цвету, если он там есть. Пусть, например, в некоторый момент времени состояние таково (стрелкой указан обрабатываемый элемент):

Массив «Цвет»	Вспомогательный массив 1	Вспомогательный массив 2
1 Голубые	Голубые	2
2 Карие	Карие	1
3 Голубые		
4 Зеленые		
5 Зеленые		
6 Голубые		

Просматриваем вспомогательный массив 1, не обнаруживаем там зеленого цвета, поэтому записываем его. Во вспомогательный массив 2 в соответствующий элемент записываем 1. Изменившееся состояние таково:

Массив «Цвет»	Вспомогательный	Вспомогательный

	массив 1	массив 2
1 Голубые	Голубые	2
2 Карие	Карие	1
3 Голубые	Зеленые	1
4 Зеленые		
5 Зеленые		
6 Голубые		

Просматриваем вспомогательный массив 1 для следующего обрабатываемого элемента, обнаруживаем искомый цвет и увеличиваем соответствующий элемент вспомогательного массива 2. Изменившееся состояние таково:

Массив «Цвет»	Вспомогательный массив 1	Вспомогательный массив 2
1 Голубые	Голубые	2
2 Карие	Карие	1
3 Голубые	Зеленые	2
4 Зеленые		
5 Зеленые		
6 Голубые		

Составлем программу:

```
program task11_59;
var     n : integer; { количество жителей Цветочного города }
        a : array [1..100] of string; { массив цветов глаз }
        b1 : array [1..100] of string; { вспомогательный массив 1 }
        b2 : array [1..100] of integer;{ вспомогательный массив 2 }
        m : integer; { количество элементов, записанных во
вспомогательных массивах }
        i,j : integer; { индексы }
        f : boolean; { логическая переменная }
begin
   write('Введите n<100 -количество жителей Цветочного города ');
   readln(n);
   for i:=1 to n do
   begin  write('Введите цвет глаз очередного жителя ',
                'Цветочного города ');
          read(a[i])
   end;
   m := 0; { нет пока элементов во вспомогательных массивах }
   for i:=1 to n do
   begin
          j := 1;
          f := false;
          while (j<=m) and not f do
                if a[i]=b1[j]
```

```
                then    begin  b2[j] := b2[j]+1;
                               f := true
                        end
                else j := j+1;
        if not f
        then    begin  m := m+1;
                       b1[m] := a[i];
                       b2[m] := 1
                end;
    end;
    { Найдем max в массиве b2 }
    j := 1; { Пусть пока это будет 1-й элемент }
    for i:=1 to m do
        if b2[j]<b2[i]
        then j := i;
    write('Наиболее часто встречается ',b1[j],' цвет глаз')
end.
```

Упражнения:

1. У каждого из n предпринимателей есть денежный капитал, размер которого они друг от друга скрывают. Все они хранят деньги в одном банке. Служащие банка имеют доступ только к следующей информации:

1) сумме капиталов всех предпринимателей, кроме первого;

2) сумме капиталов всех предпринимателей, кроме второго; и так далее;

n) сумме капиталов всех предпринимателей, кроме n-го.

Определить размер капитала каждого предпринимателя.

2. На n складах хранится товар. Известна сумма хранящегося товара на любых двух складах вместе. Нужно узнать, сколько товара хранится на каждом складе в отдельности.

3. В двумерном массиве n·n выбрать такую диагональ, параллельную главной, сумма элементов которой была бы максимальной.

4. В первой строке двумерного массива названы животные, во второй строке - города, в зоопарках которых они содержатся. Например,

Слон	Слон	Тигр
Москва	Рим	Пермь

...

Указать названия животных, которые содержатся только в одном заопарке.

5. Вычислить: s=2+22+...+222 ... 2. Последнее слагаемое содержит n цифр, где n>1997.

6. Двумерный массив n·n заполнен 0,1 и 2. Выяснить, стоят ли в нем два нуля рядом по горизонтали или вертикали.

7. Задан одномерный массив. Нужно упорядочить только отрицательные его элементы, оставив положительные на старых местах.

8. Проверить, есть ли в квадратной матрице n·n такая строка, каждый элемент которой больше суммы элементов каждой другой строки.

9. В одномерном массиве целых чисел найти максимальный среди элементов, являющихся четными, и минимальный среди элементов, кратных A.

10. Найти самый короткий путь между максимальным и минимальным элементами двумерного массива, если разрешается двигаться вправо, влево, вверх, вниз и по диагонали.

11. В двумерном массиве n·m найти среднее арифметическое первого столбца и количество элементов в каждом из следующих столбцов, превышающих среднее арифметическое предыдущего столбца.

12. РАБОТА СО СТРОКАМИ

Строки представляют собой последовательность символов используемой кодовой таблицы. Строка характеризуется своей длиной. Строку можно интерпретировать как массив, поэтому методы работы с массивами подходят и для строк. Однако для строк применяются и специфические методы, основанные на процедурах и функциях работы со строками. Эти процедуры и функции описаны нами ранее. В этой главе рассмотрим специфические методы работы со строками.

Пример 12.1. Подсчитать, сколько раз в заданной строке встречается указанная буква.

Решение 1. Обозначим заданную строку - s, а искомую букву - a. Тогда для решения задачи будем просматривать заданную строку посимвольно и каждый символ сравнивать с заданной буквой.

```
k:=0; { количество указанных букв в строке }
for i:=1 to length(s) do
    if copy(s,i,1)=a then k:=k+1.
```

Решение 2. Будем искать положение указанной буквы в строке до тех пор, пока ее удастся найти. Затем отбрасываем ту часть строки, где была найдена указанная буква, и повторяем поиск.

```
k:=0; j:=pos(a,s); { позиция первого вхождения a в строку s }
while j<>0 do
begin     k:=k+1;
          s:=copy(s,j+1,length(s)-j)
          j:=pos(a,s)
end.
```

Пример 12.2. Проверить, входят ли в строку s две буквы a.

Решение. Если бы требовалось найти две буквы a, стоящие подряд, то задача бы решалась просто: if pos('aa',s)>0 then write('входят') else write('НЕ входят'). Однако здесь требуется проверить наличие двух букв a, стоящих в любом месте строки. Эта задача является поисковой. Если строка закончится и две буквы a не будут найдены, то ответ на вопрос задачи отрицательный. Если при поиске будут найдены две буквы a, то ответ на вопрос задачи положительный. Для подсчета найденных букв a используется счетчик.

```
k:=0; { счетчик букв a }
f:=false; { пока не нашли две буквы a }
i:=1; { номер исследуемого символа строки }
while (i<=length(s)) and not f do
   if copy(s,i,1)='a'
   then   begin  k:=k+1;
                 if k=2 then f:=true
```

```
            end
    else i:=i+1;
if f
then write('в строке есть две буквы а')
else write('в строке нет двух букв а').
```

Другое решение этой задачи можно получить, основываясь на втором решении задачи 12.1.

```
j:=pos('a',s);
if j>0
then    begin  s:=copy(s,j+1,length(s)-j);
               j:=pos('a',s);
               if j>0
               then write('в строке есть две буквы а')
               else write('в строке нет двух букв а')
        end
else write('в строке нет двух букв а').
```

Упражнения:

1. Проверьте, встречаются ли в заданной строке только две буквы а.
2. Сравните решение задачи 12.2 с решением упражнения 1.
3. Найдите i-е вхождение символа x в строку s.

Пример 12.3. В строке s заменить символы а на символы я.

Решение 1. Просматриваем строку посимвольно, удаляем найденный символ а, вставляем на его место я.

```
for i:=1 to length(s) do
if copy(s,i,1)='a'
then    begin  delete(s,i,1)
               insert('я',s,i);
        end.
```

Решение 2. Просматриваем исходную строку посимвольно и переписываем в выходную строку символы, отличные от а. Вместо символа а переписываем символ я.

```
s1:=''; { выходная строка }
for i:=1 to length(s) do
    if copy(s,i,1)='a'
    then s1:=s1+'я'
    else s1:=s1+copy(s,i,1).
```

Решение 3. Оно основывается на решении 2 задачи 12.1.

```
j:=pos('a',s);
while j<>0 do
begin   s:=copy(s,1,j-1)+'я'+copy(s,j+1,length(s)-j);
        j:=pos('a',s)
end.
```

Упражнение. Замените в строке s символы аа, стоящие рядом, на символ я.

Пример 12.4. Назовем словом любую последовательность букв и цифр. Строка состоит из слов, разделенных одним или несколькими пробелами. Удалить лишние пробелы, оставив между словами по одному пробелу.

Решение. Лишними пробелами называются второй, третий и т.д., следующие за первым пробелом. Следовательно, чтобы найти лишний пробел, нужно искать два пробела, стоящие рядом, и удалять второй пробел в каждой найденной паре.

```
j:=pos(' ',s);
while j<>0 do
begin     delete(s,j,1);
          j:=pos(' ',s)
end.
```

Пример 12.5. Строка символов - это любая последовательность символов, заключенная в апострофы. Задана строка символов, состоящая из слов и строк, разделенных одним или несколькими пробелами. Нужно удалить из строки все незначащие пробелы. Незначащими пробелами называются пробелы, не стоящие в апострофах.

Решение. Запишем формально определение незначащего пробела. Текущий пробел незначащий, если предыдущий символ является пробелом и этот пробел не стоит в апострофах. Введем логическую переменную p, которая принимает значение false, если предыдущий символ не является пробелом, и значение true, если предыдущий символ - пробел. Введем логическую переменную q, которая принимает значение false, если пробел находится не в апострофах, и значение true, если пробел в апострофах. Тогда формальное определение незначащего пробела запишется так:
(copy(s,i,1)=' ') and p and not q (текущий символ пробел И предыдущий пробел И не в апострофах).

Составляем программу:

```
p:=false;
q:=false;
s1:='';
for i:=1 to length(s) do
begin     if not((copy(s,i,1)=' ') and p and not q)
          then s1:=s1+copy(s,i,1);
          if copy(s,i,1)=' '
          then p:=true else p:=false;
          if copy(s,i,1)='' then q:=not q
end;
```

Пример 12.6. Подсчитать количество гласных русских букв в строке.

Решение. Гласной буквой является такая буква, которая принадлежит множеству гласных букв. Для решения задачи просматриваем строку посимвольно и проверяем каждый символ на принадлежность гласным буквам:

```
k:=0; { количество гласных букв }
for i:=1 to length(s) do
if pos(copy(s,i,1),'аоуэыяёюеи')>0
then k:=k+1;
```

Пример 12.7. Задано предложение, состоящее из слов, разделенных одним или несколькими пробелами. Определить самое длинное слово предложения.

Решение. Чтобы выделить окончание слова, нужно анализировать два символа: первый символ должен быть отличен от пробела, а второй должен быть пробелом.

Для одинаковой обработки всех символов предложения добавим к концу предложения дополнительный символ - пробел. Как только обнаружится конец слова, вычислим его длину и проверим на максимум:

```
smax:=''; { слово максимальной длины }
readln(s); { исходное предложение }
s:=s+' '; { исходное предложение с дополнительным пробелом }
ss:='' { текущее слово предложения }
for i:=1 to length(s)-1 do { просмотр предложения по два символа }
   if (copy(s,i,1)<>' ') and (copy(s,i+1,1)=' ')
{ если текущий символ не пробел, а следующий - пробел }
   then   begin  ss:=ss+copy(s,i,1); { дописали последний символ }
                 if length(smax)<length(ss)
                 then smax:=ss; { если длина нового слова больше, чем
длина smax, то запоминаем его }
                 ss:='' { готовим место для следующего слова }
          end
   else if copy(s,i,1)<>' ' then ss:=ss+copy(s,i,1).
{ если текущий символ не пробел, то запоминаем его в слове }.
```

Пример 12.8. Задано предложение, состоящее из слов, разделенных одним или несколькими пробелами. Упорядочить слова предложения в алфавитном порядке.

Решение. Перепишем слова предложения по одному в элементы одномерного массива. Отсортируем массив по возрастанию и перепишем слова из массива в строку.

```
const nn=100; { максимальное количество слов в предложении }
type mas=array[1..nn]of string; { тип массива строк }
var      a:mas; { массив слов предложения }
         i,j, { индексы массивов, i - номер обрабатываемого символа }
         k:integer; { количество слов в предложении, индекс массива слов }
         s,  { исходное предложение и результат }
         r:string;{ текущее слово предложения, переменная для обмена слов }
begin
   write('Введите строку ');
   readln(s);
   s:=s+' '; { добавили пробел в конце для однотипной обработки всех слов }
   k:=0; { количество слов в предложении }
   r:=''; { текущее слово }
   for i:=1 to length(s)-1 do {просмотр предложения по два символа}
         if (copy(s,i,1)<>' ') and (copy(s,i+1,1)=' ')
         { если текущий символ не пробел, а следующий пробел }
         then   begin  r:=r+copy(s,i,1); { дописать символ к слову }
                       k:=k+1; { увеличить количество слов }
                       a[k]:=r; { записать слово в массив}
                       r:='' { подготовить место для следующего слова }
                end
         else if copy(s,i,1)<>' ' then r:=r+copy(s,i,1);
   { если текущий символ не пробел, то записать его в слово }
```

```
    { три следующих оператора сортируют массив }
    for i:=1 to k-1 do
    for j:=i+1 to k do
            if a[i]>a[j] then begin r:=a[i];a[i]:=a[j];a[j]:=r end;
    { сцепление слов из массива в новое предложение }
    s:='';
    for i:=1 to k do s:=s+a[i]+' ';
    write(s);
end.
```

13. ПРОЦЕДУРЫ И ФУНКЦИИ

13.1. Общие понятия

Процедуры и функции позволяют оформить часто используемый алгоритм в виде, удобном для его многократного исполнения. Процедура может возвращать в качестве ответа несколько значений, а функция только одно. Работа с процедурами и функциями состоит из двух частей:

1) описания процедуры или функции;

2) вызова ее на исполнение (передача управления компьютером) с одновременной передачей исходных данных, необходимых для работы процедуры или функции.

По окончании работы процедуры или функции управление возвращается за точку вызова(к следующему оператору).

Описание процедуры:

```
procedure имя [(список параметров)];
Описание данных;
begin
        Операторы процедуры
end;
```

Список параметров предназначен для указания типа, количества и порядка следования исходных данных и результатов. При вызове процедуры или функции по определенным правилам будут подставляться аргументы, указанные при вызове. Параметры бывают двух видов: *параметры-значения и параметры-переменные*. На место параметра-значения подставляется значение выражения, передаваемого как аргумент. Этот параметр описывается указанием имени и типа. Его значение не может быть возвращено в точку вызова. На место параметра-переменной подставляется переменная, ее значение может быть изменено в процедуре или функции и возвращено обратно. Описанию этого параметра должно предшествовать ключевое слово var.

Процедура может не иметь параметров. В этом случае информация передается в процедуру и обратно с помощью глобальных данных, т.е. данных, описанных в главной программе и не объявленных снова в процедуре.

Переменные, описанные в процедуре, обладают свойством локальности. Их областью действия является только та процедура или функция, где они описаны, а также те, вложенные процедуры и/или функции, в которых переменные с такими же именами не объявлены заново.

Вызов процедуры осуществляется оператором: *имя[(список аргументов)]*. В момент вызова между списком параметров и списком аргументов устанавливается соответствие по количеству, типу и порядку следования.

Описание функции:

function имяф(список_параметров) : тип_возвращаемого_значения;
Описание данных;
begin
Операторы функции;
имяф := выражение; {обязательный оператор, формирующий возвращаемое функцией значение}
end;

Вызов функции осуществляется с помощью указателя функции, который записывается в каком-либо выражении: *имяф(список_аргументов)*.

Пример 13.1. Вычислить $s = 1 + x/1! + x^2/2! + x^3/3! + ... + x^n/n!$.

Решение. Вычисление степени, факториала и суммы оформим в виде функций:

```
program task1;
var     x : real; {аргумент}
        n : integer; {количество слагаемых в сумме}
        y : real; {сумма}

{функция вычисления числителя}
function a(x : real; i : integer) :real;
var j : integer; {счетчик умножений}
begin   a := 1;
        for j := 1 to i do a := a * x
end;

{функция вычисления знаменателя}
function b(i : integer) :real;
var j : integer; {счетчик умножений}
begin   b := 1;
        for j := 1 to i do b := b * j
end;

{функция вычисления суммы}
function y(x : real; n : integer) : real;
var i : integer; {номер очередного слагаемого}
begin   y := 0;
        for i := 1 to n do y := y + a(x,i)/b(i)
end;

{основная программа}
begin
   write('введите аргумент - x, и количество слагаемых - n ');
   readln(x,n);
   write ('Сумма равна ',y(x,n))
```

```
end.
```

Пример 13.2. Найти наименьший член последовательности a(n)=n·n-7·n+1, где n изменяется от 1 до m.

Решение. Для поиска минимального элемента используем функцию, находящую минимальный элемент из двух параметров:

```
program task2;
var     m : integer; {количество элементов последовательности}
        n : integer; {номер текущего элемента последовательности}
        k : real; {минимальный элемент последовательности}

function min(p1,p2 :real) :real;
begin   if p1 < p2
        then min := p1
        else min := p2
end;
begin
   write('Введите m - количество элементов последовательности');
   readln(m);
   k := 1*1-7*1+1;
   for n := 2 to m do
        k := min(k,n*n-7*n+1);
   write('Минимальный элемент последовательности равен ',k)
end.
```

Пример 13.3. Написать набор процедур для работы с обыкновенными дробями, обеспечив их сложение, вычитание, умножение, деление.

Решение. Обыкновенную дробь будем изображать двумя целыми числами: первое число будет представлять числитель дроби, а второе - знаменатель. В процессе вычислений требуется сокращать дроби на их наибольший общий делитель (НОД), для вычисления которого используется алгоритм Евклида. Если одно из чисел равно нулю, то НОД берем равным 1. Разработаем также отдельные процедуры для ввода и вывода обыкновенных дробей:

```
Var     x,y, {числитель и знаменатель дроби }
        p,q, {числитель и знаменатель дроби }
        s,t:integer; {числитель и знаменатель дроби }

{ Ввод обыкновенной дроби }
procedure wwod(var a,b:integer);
begin
   writeln;
   write('Введите целые: числитель и знаменатель обыкновенной дроби ');
   readln(a,b)
end;

{ Вывод результата }
procedure wywod(a,b:integer);
begin write(a,'/',b);writeln end;
```

```
{ Вычисление НОД(x,y) }
function nod(x,y:integer):integer;
begin     if (x=0) or (y=0)
          then nod:=1
          else    begin  while x<>y do
                         begin  while x>y do x:=x-y;
                                while y>x do y:=y-x
                         end;
                         nod:=x
                  end
end;

{ Сокращение дроби }
procedure sokr(var c,d:integer);
var       r:integer;
begin     r:=nod(c,d);
          c:=c div r;
          d:=d div r
end;


{ Сложение двух дробей }
procedure sum(a,b,c,d:integer; var e,f:integer);
var r:integer;
begin     e:=a*d+b*c;
          f:=b*d;
          sokr(e,f)
end;

{ Вычитание двух дробей }
procedure raz(a,b,c,d:integer; var e,f:integer);
var r:integer;
begin     e:=a*d-b*c;
          f:=b*d;
          sokr(e,f)
end;

{ Умножение двух дробей }
procedure mult(a,b,c,d:integer; var e,f:integer);
var r:integer;
begin     e:=a*c;
          f:=b*d;
          sokr(e,f)
end;
```

```
{ Деление двух дробей }
procedure del(a,b,c,d:integer; var e,f:integer);
var r:integer;
begin     e:=a*d;
          f:=b*c;
          sokr(e,f)
end;

begin
   write('Введите первую дробь ');
   wwod(x,y);
   write('Введите вторую дробь ');
   wwod(p,q);
   write('Сумма равна '); sum(x,y,p,q,s,t); wywod(s,t);
   write('Разность равна '); raz(x,y,p,q,s,t); wywod(s,t);
   write('Произведение равно '); mult(x,y,p,q,s,t); wywod(s,t);
   write('Частное равно '); del(x,y,p,q,s,t); wywod(s,t);
end.
```

Пример 13.4. Выпуклый многоугольник на плоскости задается количеством вершин и координатами каждой вершины, перечисленными в порядке обхода сторон многоугольника по часовой стрелке. Написать набор процедур и функций для вычисления длины стороны многоугольника, его периметра и площади, длины наибольшей стороны и диагонали.

Решение. Для представления многоугольника будем использовать целую переменную n - количество вершин, и двумерный массив из двух строк. В первой строке массива будем размещать абсциссы, а во второй строке в соответствующих элементах - ординаты вершин многоугольника. Для вычисления длины стороны воспользуемся известной формулой. Для вычисления периметра будем обходить стороны многоугольника от первой вершины к последней по часовой стрелке и суммировать длины сторон. В результате такого же обхода можно найти и длину наибольшей стороны многоугольника.

Для вычисления площади многоугольника разобьём его диагоналями, выходящими из вершины 1, на треугольники. Найдем площадь каждого треугольника и сумму этих площадей, которая и будет равна площади многоугольника. Вершина 1 здесь была выбрана произвольно, можно было выбрать любую другую вершину.

Для вычисления длины наибольшей диагонали нужно перебрать все диагонали многоугольника. При этом уже просмотренные диагонали перебирать не надо. Например, для n=6 нужно просмотреть диагонали между вершинами 13, 14, 15, 24, 25, 26, 35, 36, 46. Замечаем, что для первой точки возможные диагонали заканчиваются в точке n-1, а для остальных точек - в точке 6.

```
const nn=100; { максимальное количество вершин выпуклого
многоугольника }
type mnogoug=array[1..2,1..nn]of real; { тип массива вершин }
var       a:mnogoug; { многоугольник }
          n:integer; { количество вершин }
```

```
{ Длина стороны}
function dlina(x1,y1,x2,y2:real):real;
begin dlina:=sqrt(sqr(x1-x2)+sqr(y1-y2)) end;

{ Площадь треугольника }
function treug(x1,y1,x2,y2,x3,y3:real):real;
var     a,b,c:real;
        p:real;
begin   { вычисление длин сторон }
        a:=dlina(x1,y1,x2,y2);
        b:=dlina(x1,y1,x3,y3);
        c:=dlina(x2,y2,x3,y3);
        p:=(a+b+c)/2; { полупериметр }
        { площадь треугольника по формуле Герона }
        treug:=sqrt(p*(p-a)*(p-b)*(p-c))
end;

{ Периметр многоугольника }
function perim(a:mnogoug;n:integer):real;
var     i:integer; { номера вершин }
        d:real; { периметр }
begin   { длина стороны между вершиной 1 и вершиной n }
        d:=dlina(a[1,1],a[2,1],a[1,n],a[2,n]);
        { обход сторон и вычисление суммы их длин }
        for i:=1 to n-1 do
                d:=d+dlina(a[1,i],a[2,i],a[1,i+1],a[2,i+1]);
        perim:=d
end;

{ Площадь многоугольника }
function area(a:mnogoug;n:integer):real;
var     d:real; { площадь }
        i:integer; { номера вершин }
begin   d:=0;
{ Разбиваем многоугольник на треугольники диагоналями, проведенными из
вершины 1 }
        for i:=2 to n-1 do
                d:=d+treug(a[1,1],a[2,1],a[1,i],a[2,i],a[1,i+1],a[2,i+1]);
        area:=d
end;

{ Длина наибольшей стороны многоугольника }
function stor(a:mnogoug;n:integer):real;
var     d,  { наибольшая сторона }
        d1:real; { текущая сторона }
```

```
            i:integer; { номер вершины }
    begin
        d:=dlina(a[1,1],a[2,1],a[1,n],a[2,n]);
        for i:=1 to n-1 do
        begin  d1:=dlina(a[1,i],a[2,i],a[1,i+1],a[2,i+1]);
            if d<d1 then d:=d1
            end;
        stor:=d
    end;

    { Длина наибольшей диагонали многоугольника }
    function diag(a:mnogoug;n:integer):real;
    var     d,  { наибольшая диагональ }
            d1:real; { текущая диагональ }
            i,  { номер второй вершины диагонали }
            j,  { номер первой вершины диагонали }
            m:integer; { номер последней вершины многоугольника куда можно
провести диагональ из вершины j }
    begin
        d:=0;
        for j:=1 to n-2 do { n-2 номер последней вершины откуда еще можно
проводить непросмотренные диагонали}
        begin
            if j=1 then m:=n-1 else m:=n; { определение номера последней
вершины куда можно провести диагональ из вершины j }
            for i:=j+2 to m do
            begin  d1:=dlina(a[1,j],a[2,j],a[1,i],a[2,i]);
                if d<d1 then d:=d1;
                writeln(j,i)
            end
        end;
        diag:=d
    end;

    { Ввод многоугольника }
    procedure wwod(var a:mnogoug;var n:integer);
    var     i:integer;
    begin   write('Введите количество вершин многоугольника ');
            readln(n);
            for i:=1 to n do
            begin  write('Введите координаты ',i,' вершины ');
                readln(a[1,i],a[2,i])
            end
    end;

    begin
```

```
    wwod(a,n);
    writeln('Периметр многоугольника ',perim(a,n));
    writeln('Площадь многоугольника ',area(a,n));
    writeln('Длина наибольшей стороны ',stor(a,n));
    writeln('Длина наибольшей диагонали ',diag(a,n));
end.
```

13.2. Рекурсивные процедуры и функции

В общем случае *рекурсией* называется ситуация, когда какой-то алгоритм вызывает себя прямо или через другие алгоритмы в качестве вспомогательного. Сам алгоритм при этом называется рекурсивным. Понятно, что без конца такие вызовы продолжаться не могут, так как в противном случае получится бесконечный цикл. Поэтому при построении рекурсивного алгоритма предусматриваются случаи, когда результат вычисляется явно (непосредственно) без самовызова.

Таким образом, любая рекурсия обязательно должна содержать два условия:

1) Вычисление результата через другие значения (для простейших случаев). Выполнение этого условия не должно повлечь за собой нового рекурсивного вызова.

2) Вычисление значения с помощью самовызова функции (рекурсивный вызов).

Пример 13.5. Широко известно рекурсивное определение факториала:

$$n = \begin{cases} 1, & \text{если } n = 0 \text{ или } n = 1 \\ (n-1)!, & \text{если } n > 1. \end{cases}$$

Здесь n - неотрицательно. Запишите эту функцию на Паскале.

Решение. В первой строке определения явно указано, как вычислить факториал, если аргумент равен нулю или единице. В любом другом случае для вычисления n! необходимо вычислить предыдущее значение (n-1)! и умножить его на n. Уменьшающееся значение гарантирует, что в конце концов возникнет необходимость найти 1! или 0!, которые вычисляются непосредственно.

```
program task5;
var n : integer; { исходное значение }

function fact(i:integer):integer;
begin     if (i=1) or (i=0)
          then fact := 1
          else fact := fact(i-1)*i
end;

begin     write('Введите нужное значение n ');
          readln(n);
          writeln('Факториал ',n,' равен ',fact(n))
end.
```

Чтобы понять, как будет выполняться эта программа, вспомним, что на время выполнения вспомогательного алгоритма основной алгоритм приостанавливается. При вызове новой копии рекурсивного алгоритма вновь выделяется место для всех

переменных, объявляемых в нем, причем переменные других копий будут недоступны. При удалении копии рекурсивного алгоритма из памяти удаляются и все его переменные. Активизируется предыдущая копия рекурсивного алгоритма, становятся доступными ее переменные. Пусть необходимо вычислить 4! Основной алгоритм: вводится n=4, вызов fact(4). Основной алгоритм приостанавливается, вызывается и работает fact(4): 4<>1 и 4<>0, поэтому fact:=fact(3)*4. Работа функции приостанавливается, вызывается и работает fact(3): 3<>1 и 3<>0, поэтому fact:=fact(2)*3. Заметьте, что в данный момент в памяти компьютера две копии функции fact. Вызывается и работает fact(2): 2<>1 и 2<>0, поэтому fact:=fact(1)*2. В памяти компьютера уже три копии функции fact и вызывается четвертая. Вызывается и работает fact(1): 1=1, поэтому fact(1)=1. Работа этой функции завершена, продолжает работу fact(2). fact(2):=fact(1)*2 =1*2=2. Работа этой функции также завершена, и продолжает работу функция fact(3). fact(3):=fact(2)*3=2*3=6. Завершается работа и этой функции, и продолжает работу функция fact(4). fact(4):=fact(3)*4= 6*4=24. Сейчас управление передается в основную программу и печатается ответ: «Факториал 4 равен 24».

Приведем еще несколько рекурсивных алгоритмов.

Пример 13.6. Написать рекурсивную функцию, вычисляющую указанное число Фибоначчи.

Решение. Последовательность Фибоначчи задается следующими соотношениями: a(0)=a(1)=1, a(i)=a(i-1)+a(i-2), где i>1, которые легко записать на Паскале в виде рекурсивной функции.

```
function fib(n:integer):integer;
begin    if n=0
         then fib := 1
         else   if n=1
                then fib := 1
                else fib := fib(n-1)+fib(n-2)
end;
```

Пример 13.7. Написать рекурсивную функцию для вычисления квадрата натурального числа.

Решение. Известно, что $(n+1)^2=n^2+2*n+1$ и $1^2=1$, отсюда

$$\text{Œ}(n) = \begin{cases} 1, & \text{æ}^{\circ}\text{Ł } n = 1, \\ \text{Œ}(n-1) + 2\cdot(n-1) + 1, & \text{æ}^{\circ}\text{Ł } n > 1. \end{cases}$$

Отсюда

```
function kv(i:integer):integer;
begin    if i=1
         then kv := 1
         else kv := kv(i-1)+2*(i-1)+1
end;
```

Пример 13.8. Написать рекурсивную функцию сложения целых чисел.

Решение. В силу законов ассоциативности и коммутативности имеем (a+1)+(b-1)=a+b. Если b=0, то сумма совпадает с a.

Поэтому получаем следующую функцию:

```
function plus(a,b:integer):integer;
begin    if b=0
```

```
            then plus := a
            else    if b>0
                    then plus := plus(succ(a),pred(b))
                    else plus := plus(pred(a),succ(b))
end;
```

Пример 13.9. Написать рекурсивную процедуру, которая считывает вводимые с клавиатуры числа до тех пор, пока не будет обнаружен ноль. Затем введенные числа распечатываются в обратном порядке.

Решение.

```
procedure wri;
var a:real;
begin       read(a);
            if a<>0 then wri;
            write(a,' ')
end;
```

Упражнение. Измените процедуру примера 13.9 так, чтобы ноль не печатался.

Пример 13.10. Написать рекурсивную процедуру, переводящую целое число из десятичной системы счисления в восьмиричную.

Решение.

```
procedure convert(z:integer);
begin       if z>1 then convert(z div 8);
            write(z mod 8:1)
end;
```

Упражнение. Исправьте процедуру примера 13.10 так, чтобы можно было перевести число в шестнадцатиричную систему счисления.

Пример 13.11. Написать рекурсивную функцию для поиска максимального элемента в одномерном массиве.

Решение. Предположим, что массив описан в следующем операторе описания типа: type mas= array [1..n] of real;

```
function m(a:mas;i:integer):real;
begin       if i=1
            then m := a[1]
            else    if a[i]>m(a,i-1)
                    then m := a[i]
                    else m := m(a,i-1)
end;
```

Пример 13.12. Задан прямоугольник со сторонами A и B (A,B - натуральные числа). Разбиваем его на части с помощью квадратов. Определить, сколько квадратов получится, если каждый раз выбирается самый большой квадрат.

Решение. Стороны квадрата равны. Наибольший квадрат получается, если от большей стороны отнять меньшую. Продолжаем эту операцию до тех пор, пока длины сторон не совпадут. Обозначим длину большей стороны - y, а длину меньшей стороны - x.

```
{ глобальные переменные }
var         y:integer; { длина большей стороны прямоугольника }
```

```
        x:integer; { длина меньшей стороны прямоугольника }
function min(i,j:integer):integer;
begin if i<j then min:=i else min:=j end;
function max(i,j:integer):integer;
begin if i>j then max:=i else max:=j end;
function f:integer;
var d:integer; { длина стороны отсекаемого квадрата }
begin   if x=y
        then f:=1
        else    begin  d:=y-x;
                       y:=max(d,x);
                       x:=min(d,x);
                       f:=f+1
                end
end.
```

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Каймин В.А. и др. Основы информатики и вычислительной техники: Пробное учебное пособие для 10-11 классов средней школы. - М.: Просвещение, 1990.
2. Гейн А.Г., Житомирский В.Г. и др. Основы информатики и вычислительной техники: Пробное учебное пособие. - Свердловск.: Изд-во Уральского университета, 1992.
3. Кушниренко А.Г. и др. Основы информатики и вычислительной техники: Пробный учебник для средних учебных заведений. - М.: Просвещение, 1992.
4. Ершов А., Кушниренко А. и др. Основы информатики и вычислительной техники: Пробный учебник для средних учебных заведений. - М.: Просвещение, 1988.
5. Каймин В.А., Жданович В.С. Информатика: Учебное пособие для старшеклассников и абитуриентов. - М.: АСТ, 1996.
6. Сенокосов А.И., Гейн А.Г. Информатика: Учебное пособие для 8-9 классов школ с углубленным изучением информатики. - М.: Просвещение, 1995.
7. Гладков В.П. Задачи по информатике на вступительном экзамене в вуз и их решения. - Пермь, Перм. гос. техн. ун-т, 1997.
8. Гладков В.П. Информатика: Учебное пособие для заочников. - Пермь, Перм. гос. техн. ун-т, 1995.
9. Фаронов В.В. Турбо Паскаль 7.0. Начальный курс: Учебное пособие. - М.: Норидж, 1997.
10. Вирт Н. Алгоритмы + структуры данных = программы. - М.: Мир, 1985.
11. Грогоно П. Программирование на языке Паскаль. - М.: Мир, 1982.
12. Пильщиков В.Н. Сборник упражнений по языку Паскаль. - М.: Наука, 1989.
13. Зелковиц М., Шоу А., Гэннон Дж. Принципы разработки программного обеспечения. - М.: Мир, 1982.
14. Майерс Г. Надежность программного обеспечения. - М.: Мир, 1980.
15. Тассел Д. Ван. Стиль, разработка, эффективность, отладка и испытание программ. - М.: Мир, 1985.

16. Йордан Э. Структурное программирование и проектирование программ. - М.: Мир, 1979.
17. Штернберг Л.Ф. Разработка и отладка программ. - М.: Радио и связь, 1984.
18. Мейер Б., Бодуэн К. Методы программирования. В 2-х томах. М.: Мир, 1982.
19. Хьюз Дж., Мичтом Дж. Структурный подход к программированию. - М.: Мир, 1980.
20. Алагич С., Арбиб М. Проектирование корректных структурированных программ. - М.: Радио и связь, 1984.
21. Кергаль И. Методы программирования на Бейсике (с упражнениями). - М.: Мир, 1991.
22. Фридман Ф., Кофман Э. Решение задач и структурное программирование на Фортране. - М.: Машиностроение, 1983.
23. Мануйлов В.Г. Разработка программного обеспечения на Паскале. - М.: Приор, 1996.
24. Грис Д. Наука программирования. - М.: Мир, 1984.
25. Гудман С., Хидетниеми С. Введение в разработку и анализ алгоритмов. - М.: Мир, 1981.
26. Зиглер К. Методы проектирования программных систем. - М.: Мир, 1985.