Министерство образования Республики Беларусь

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Кафедра интеллектуальных информационных технологий

Ю.Г. ПРИХОДЬКО, С.А. САМОДУМКИН, О.Е. ЕЛИСЕЕВА

ЛАБОРАТОРНЫЙ ПРАКТИКУМ

по курсу

ОСНОВЫ АЛГОРИТМИЗАЦИИ И ПРОГРАММИРОВАНИЯ

для студентов специальности

"Искусственный интеллект"

В 3-х частях

ЧАСТЬ 1

Минск 2000

УДК 681.3 (075.8)
ББК 32.973 Я 73
П 77

Приходько Ю.Г. и др.

Лабораторный практикум по курсу "Основы алгоритмизации и
П 77 программирования" для студентов специальности "Искусственный интеллект" /
Ю.Г. Приходько, С.А. Самодумкин, О.Е. Елисеева. В 3ч. Ч.1.–Мн.:БГУИР,2000. - с.
ISBN

В работе в первом разделе приводятся сведения и методические указания по составлению схем алгоритмов и записи алгоритмов для машины Поста. Второй раздел посвящен основам программирования на языке СИ. Практикум содержит описание заданий к лабораторным работам по курсу “Основы алгоритмизации и программирования”, который читается студентам специальности “Искусственный интеллект”.

УДК 681.3 (075.8)
ББК 32.973 Я 73

ISBN

СОДЕРЖАНИЕ

РАЗДЕЛ 1. ОСНОВЫ АЛГОРИТМИЗАЦИИ

ЛАБОРАТОРНАЯ РАБОТА № 1

Тема : схемы алгоритмов.

Цель : научиться составлять схемы алгоритмов для различных задач.

Краткие теоретические сведения

В основе решения любой задачи лежит понятие алгоритма. Под алгоритмом принято понимать “точное предписание, определяющее вычислительный процесс, ведущий от варьируемых начальных данных к искомым результатам”. Таким образом, алгоритм должен содержать конечную последовательность шагов или операций, однозначно определяющих процесс переработки исходных и промежуточных данных в искомый результат.

1.1. Алгоритм и его свойства

При составлении алгоритмов следует учитывать ряд требований, выполнение которых приводит к формированию необходимых свойств.

1. Алгоритм должен быть однозначным, исключающим произвольность толкования любого из предписаний, и заданного порядка исполнения. Это свойство алгоритма называется определённостью.

2. Реализация вычислительного процесса, предусмотренного алгоритмом, должна через определённое число шагов привести к выдаче результатов или сообщения о невозможности решения задачи. Это свойство алгоритма называется результативностью.

3. Решение однотипных задач с различными исходными данными можно осуществлять по одному и тому же алгоритму, что даёт возможность создавать типовые программы для решения задач при различных вариантах задания значений исходных данных. Это свойство алгоритма называется массовостью.

4. Предопределённый алгоритмом вычислительный процесс можно расчленить на отдельные этапы, элементарные операции. Это свойство алгоритма называется дискретностью.

Если алгоритм рассматривать как совокупность предписаний по выполнению действий, то всегда необходимо выделить те объекты, над

которыми должны осуществляться предписанные действия. Таковыми объектами являются данные.

1.2. Способы описания алгоритмов

Для строгого задания различных структур данных и алгоритмов их обработки требуется иметь такую систему формальных обозначений и правил, чтобы смысл всякого используемого предписания трактовался точно и однозначно. Соответствующие системы правил называют языками описаний.

К изобразительным средствам описания алгоритмов относятся следующие основные способы их представления: словесный (записи на естественном языке), структурно-стилизованный (записи на алгоритмическом языке псевдокода), графический (изображения схем из графических символов), в виде программы (тексты на языках программирования).

Описание алгоритма удобно представлять в виде схемы. При выполнении схем алгоритмов или программ отдельные функции алгоритмов отображают в виде условных обозначений – символов по ГОСТ 19.701-90 (ИСО 5807-85).

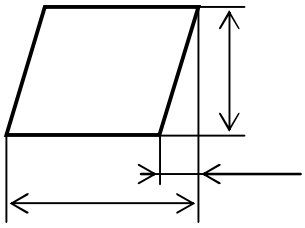
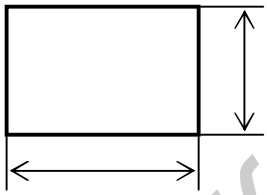
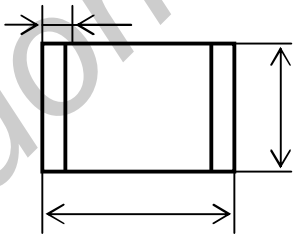
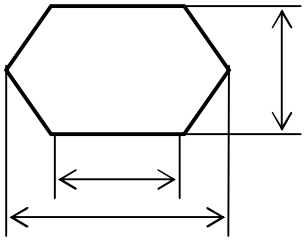
Схема - графическое представление определения, анализа или метода решения задачи, в котором используются символы для отображения операций, данных и т.д. Схема программы отображает последовательность операций в программе и состоит из:

- символов процесса, указывающих фактические операции обработки данных;
- линейных символов, указывающих поток управления;
- специальных символов, используемых для облегчения написания и чтения схем.

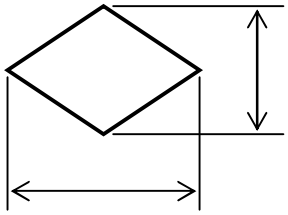
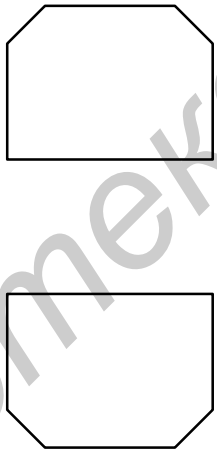
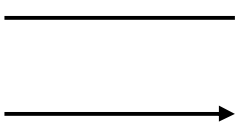
Некоторые наиболее употребительные символы приведены в табл. 1.1.

Таблица 1.1.

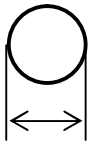
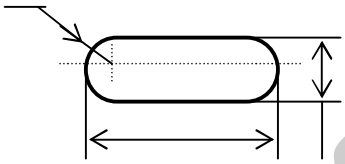
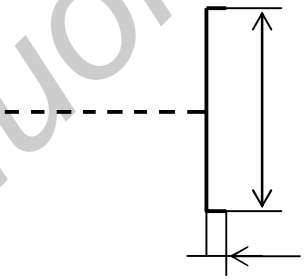
Применение символов

Название символа	Обозначение символа	Функция символа
1	2	3
Данные		Символ отображает данные. Носитель данных не определён
Процесс		Символ отображает функцию обработки данных любого вида (выполнение определенной операции или группы операций, приводящее к изменению значения, формы или размещения информации)
Предопределённый процесс		Символ отображает предопределённый процесс, состоящий из одной или нескольких операций или шагов программы, которые определены в другом листе (в подпрограмме, модуле)
Подготовка		Символ отображает модификацию команды или группы команд с целью воздействия на некоторую последующую функцию (модификация индексного регистра)

Продолжение табл.1.1

1	2	3
Решение		Символ отображает решение или функцию переключательного типа, имеющую один вход и ряд альтернативных выходов, один и только один из которых может быть активизирован после вычисления условий определенных внутри этого символа
Граница цикла		Символ, состоящий из двух частей, отображает начало и конец цикла. Обе части символа имеют один и тот же идентификатор. Условия для инициализации, приращения, завершения и т.д. помещаются внутри символа в начале или конце в зависимости от расположения операции, проверяющей условие
Линия		Символ отображает поток данных или управление. При необходимости или для повышения удобочитаемости могут быть добавлены стрелки–указатели

Окончание табл.1.1

1	2	3
Соединитель		Символ отображает выход в часть схемы и вход из другой части этой схемы и используется для обрыва линии и продолжения ее в другом месте. Соответствующие символы – соединители должны содержать одно и то же уникальное обозначение
Терминатор		Символ отображает выход во внешнюю среду и вход из внешней среды (начало или конец схемы программы, внешнее использование и источник или пункт назначения данных)
Комментарий		Символ используется для добавления описательных комментариев или пояснительных записей в целях объяснения или примечаний. Пунктирные линии в символе-комментарии связаны с соответствующим символом или могут отводить группу символов. Текст комментариев или примечаний должен быть помещен около ограничивающей фигуры

Размер *a* должен выбираться из ряда 10, 15, 20 мм. Допускается увеличивать размер *a* на число, кратное 5. Размер *b* равен 1,5*a*. При ручном выполнении схем допускается устанавливать *b* равным 2*a*.

1.3. Правила применения символов и выполнения схем

1.3.1. Правила применения символов

Символ предназначен для графической идентификации функции, которую он отображает, независимо от текста внутри этого символа.

Для облегчения вычерчивания и нахождения на схеме символов рекомендуется поле листа разбивать на зоны. Размеры зон устанавливают с учетом минимальных размеров символов, изображенных на данном листе. Координаты зоны проставляют: по горизонтали - арабскими цифрами слева направо в верхней части листа; по вертикали - прописными буквами латинского алфавита (за исключением букв I и O) сверху вниз в левой части листа.

Координаты зон в виде сочетания букв и цифр присваивают символам, вписанным в поля этих зон, например, A1, A2, A3, B1, B2, B3 и т.д., и проставляются в верхней части символа в разрыве его контура.

Символы в схеме должны быть расположены равномерно. Следует придерживаться разумной длины соединений и минимального числа длинных линий.

Большинство символов задумано так, чтобы дать возможность включить текст внутри символа. Формы символов, установленные стандартом, должны служить руководством для фактически используемых символов. Не должны изменяться углы и другие параметры, влияющие на соответствующую форму символов. Символы должны быть, по возможности, одного размера.

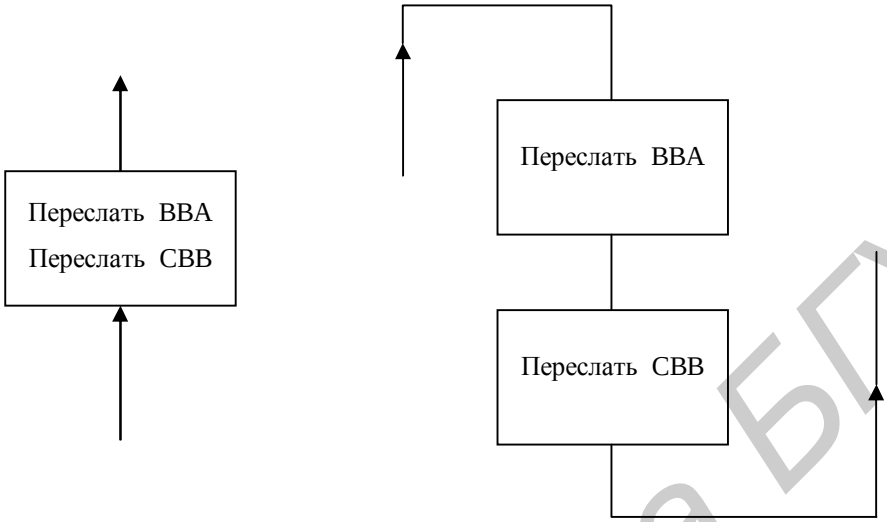
Символы могут быть вычерчены в любой ориентации, но, по возможности, предпочтительной является горизонтальная ориентация. Зеркальное изображение символа обозначает одну и ту же, но не является предпочтительной.

Расстояние между параллельными линиями потока должно быть не менее 3 мм, между остальными символами схемы - не менее 5 мм.

Минимальное количество текста, необходимое для понимания функции данного символа, следует помещать внутри данного символа. Сокращения слов

и аббревиатуры, за исключением установленных государственными стандартами, должны быть расшифрованы в нижней части поля схемы или в документе, к которому эта схема относится. Текст для чтения должен записываться слева направо и сверху вниз, независимо от направления потока.

Пример.



Если объем текста, помещаемого внутри символа, превышает его размеры, следует использовать символ комментария.

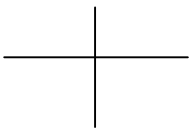
1.3.2. Правила выполнения соединений

Потоки данных или потоки управления в схемах показываются линиями.

Направления линии потока сверху вниз и слева направо принимают за основные и, если линии потока не имеют изломов, стрелками можно не обозначать. В остальных случаях, когда необходимо внести большую ясность в схему (например, при соединениях), на линиях используются стрелки.

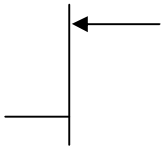
В схемах следует избегать пересечения линий. Пересекающиеся линии не имеют логической связи между собой, поэтому изменения направления в точках пересечения не допускаются.

Пример.



Две или более входящие линии могут объединяться в одну исходящую линию. Если две или более линии объединяются в одну линию, место объединения должно быть смещено.

Пример.



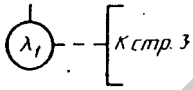
Линии в схемах должны подходить к символу слева, либо сверху, а исходить либо справа, либо снизу. Линии должны быть направлены к центру символа.

При необходимости линии в схемах следует разрывать для избежания излишних пересечений или слишком длинных линий, а также, если схема состоит из нескольких страниц. Соединитель в начале разрыва называется внешним соединителем, а соединитель в конце разрыва – внутренним соединителем.

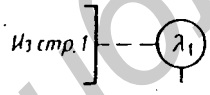
Ссылки к страницам могут быть приведены совместно с символом комментария для соединителей.

Пример.

Внешний соединитель



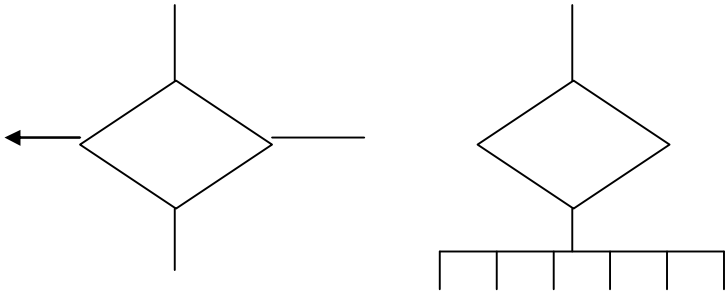
Внутренний соединитель



Несколько выходов из символа следует показывать:

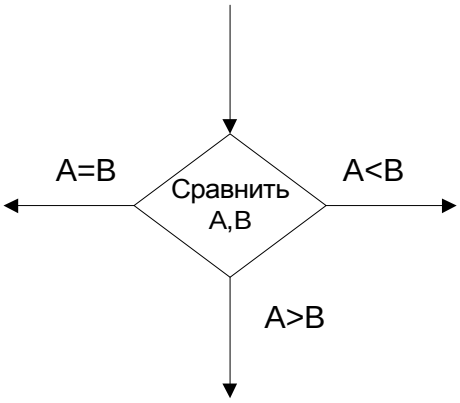
1) несколькими линиями от данного символа к другим символам;

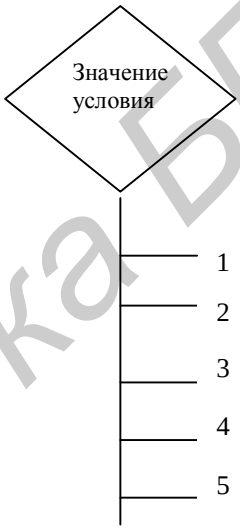
2) одной линией от данного символа, которая затем разветвляется в соответствующее число линий.



Каждый выход из символа должен сопровождаться соответствующими значениями условий, чтобы показать логический путь, который он представляет, с тем, чтобы эти условия и соответствующие ссылки были идентифицированы.

Примеры.





1.4. Выбор алгоритма

Важнейшим шагом для получения эффективной и правильной программы является составление алгоритма. Хороший алгоритм—необходимое, но не достаточное условие хорошей программы. Если пользователь формулирует задачу в виде четкого алгоритма, то процесс проектирования существенно облегчается. Проиллюстрируем это на задаче определения: является ли число *N* простым. Простое число—это число, которое делится без остатка только на единицу и на само себя. Например, 3, 5, 7,11, 13 и 17—простые числа, а числа 4, 9, 21 и 35 не являются простыми.

На рис.1.1 показана блок-схема одного из алгоритмов определения того,

является ли число *N (N>1)* простым. В соответствии с этим алгоритмом число *N* делят на 2, 3, 4,..., *N*—1 до тех пор, пока *N* не разделится без остатка или пока в качестве делителя не будет испытано число *N*. Это не самый эффективный путь решения данной задачи, но наиболее очевидный (см. рис.1.1).

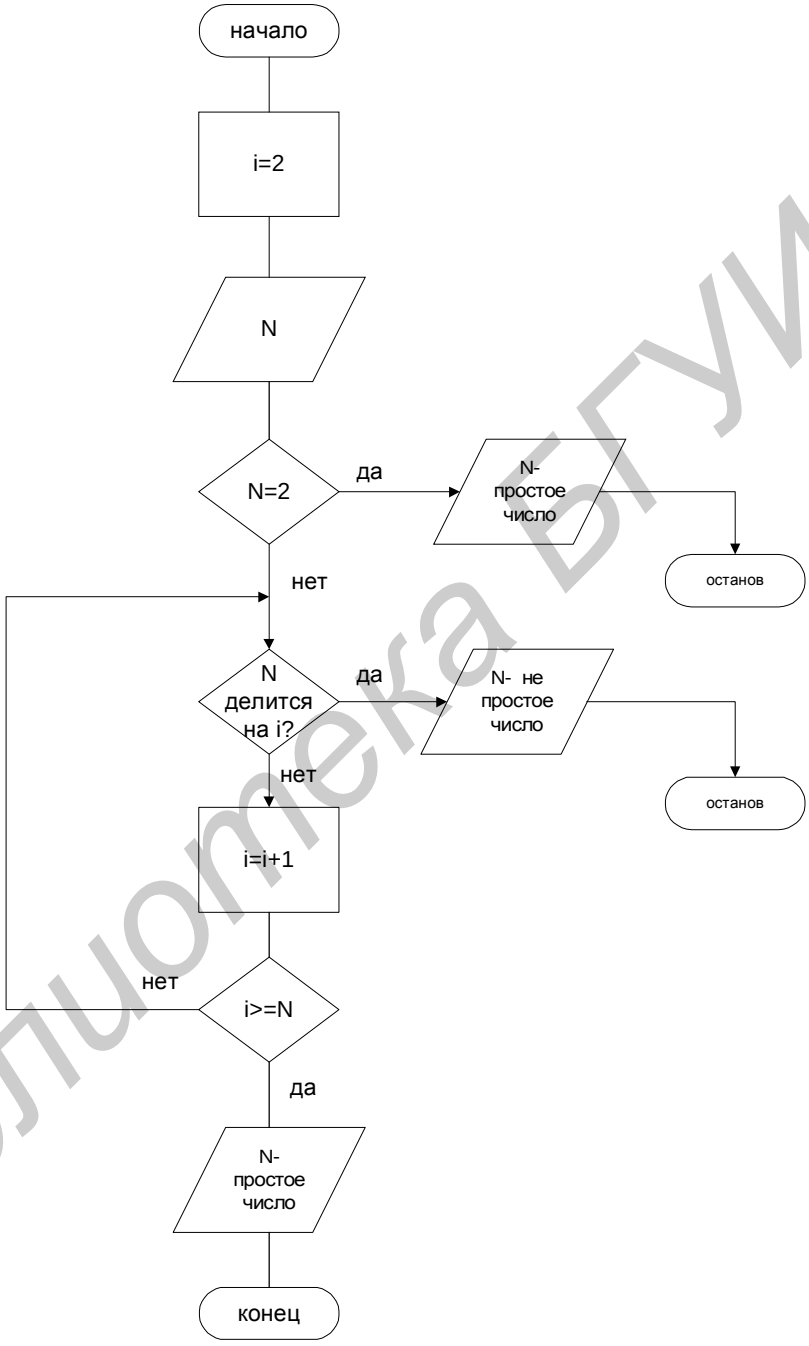


Рис. 1.1 Алгоритм определения простого числа. Вариант 1.

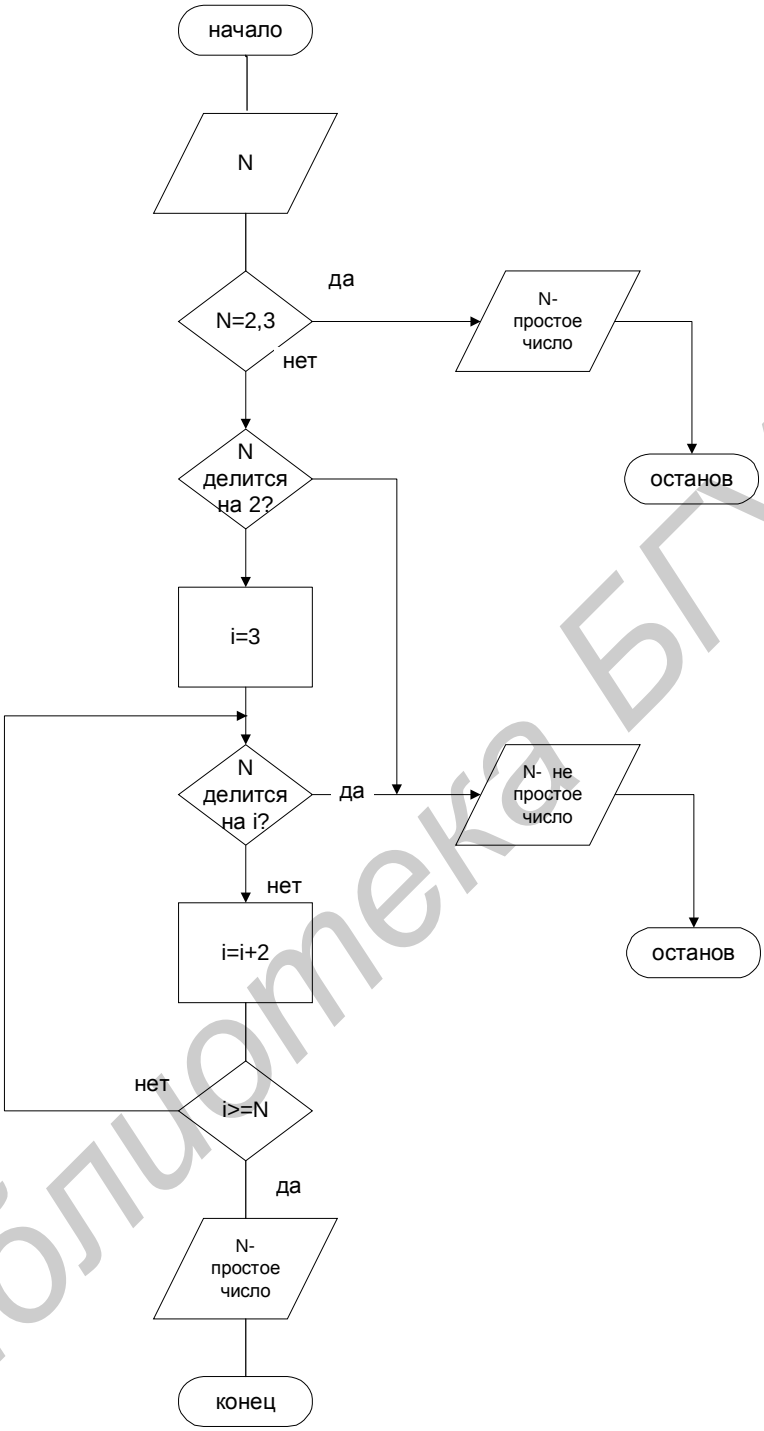


Рис. 1.2 Алгоритм определения простого числа. Вариант 2.

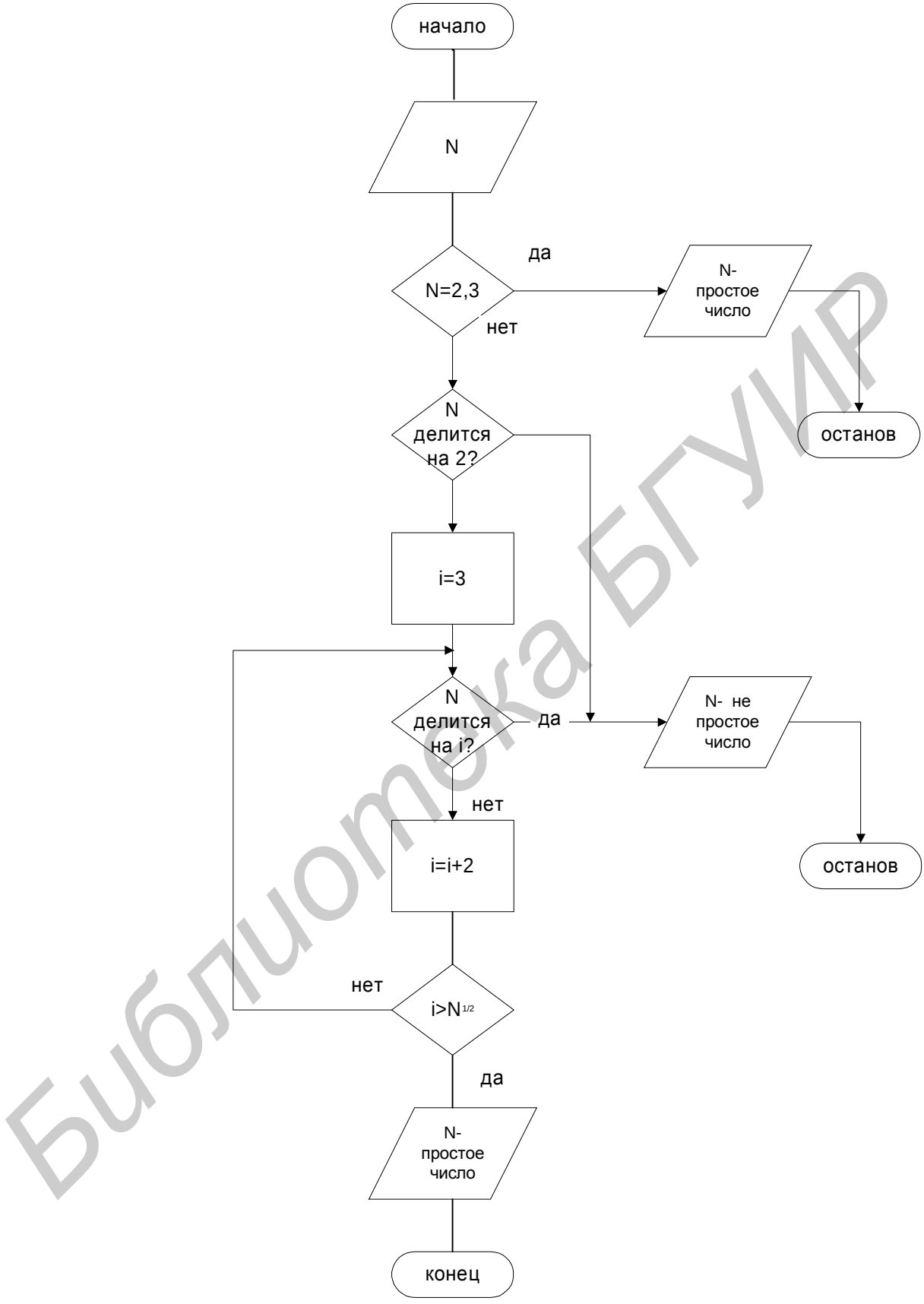


Рис. 1.3 Алгоритм определения простого числа. Вариант 3.

Нетрудно сообразить, что нет необходимости проверять, являются ли все четные числа, меньшие N, его делителями. Достаточно проверить число 2, так как 2—делитель, если любое другое четное число является делителем. Значит, надо проверять только нечетные числа 3, 5, 7 и т.д. Это уменьшит объем вычислений почти наполовину. Тогда можно составить новую блок-схему алгоритма (рис. 1.2). Этот алгоритм делит N на 2 и на все нечетные числа до тех пор, пока N не разделится на одно из них или не будет достигнуто N. Большинство программистов, освободившись почти от половины вычислений, сразу же приступят к программированию.

Но мы, чтобы подчеркнуть важность тщательного выбора алгоритма до начала программирования, проведем дальнейший анализ.

Заметим числа, меньшие или равные квадратному корню из N. Если существует делитель, больший корня квадратного из N, то должен существовать и делитель, меньший квадратного корня из N. Если это утверждение не является для вас очевидным, разберите несколько примеров. Таким образом, мы получим еще одну блок-схему, показанную на рис. 1.3. Этот алгоритм делит N на 2 и нечетные числа, меньшие или равные квадратному корню из N, до тех пор, пока N не разделится без остатка или пока не будет испытано в качестве делителя число N (см. рис.1.3).

Подсчитаем количество чисел, которое мы должны проверить в качестве делителя в каждом из трех алгоритмов (табл. 1.2).

Сравнивая второй и четвертый элементы последней строки таблицы, мы видим, что первый алгоритм проверяет более чем в 60 раз больше чисел, чем третий. С точки зрения эффективности выбор правильного алгоритма—наиболее ответственный шаг.

Таблица 1.2

Сравнение эффективности алгоритмов

	Алгоритм 1	Алгоритм 2	Алгоритм 3
N	Проверке подлежат		
	все числа <N	2 и все нечетные <N	2 и все нечетные < N
10	8	5	2
100	98	50	5
1000	998	500	16

Теперь мы имеем эффективный алгоритм, определяющий, является ли данное число простым. Пусть нам надо не определить, является ли данное число простым, а сгенерировать все простые числа. В этом случае рассмотренный алгоритм был бы неэффективным. Составить алгоритм решения такой задачи можно, если использовать решето Эратосфена.

Возникает вопрос: как выбрать хороший алгоритм? Первое правило—не начинайте программировать первый пришедший в голову алгоритм, просмотрите по крайней мере несколько вариантов. Выберите лучший из них. Если вы рассмотрели только один алгоритм для решения задачи, вряд ли он будет наилучшим.

Выбирайте алгоритм задачи самым тщательным образом.

Имеется немало литературных источников, где рассматриваются вычислительные алгоритмы. В трехтомнике Д. Кнута [3] содержится большое количество основных вычислительных алгоритмов. Все профессиональные программисты должны быть знакомы с этим полезным изданием. Кроме того, ознакомьтесь с литературными источниками, где рассматривается ваша задача, подлежащая программированию. Возможно, вы обнаружите свой алгоритм в готовых программах своих коллег. Если алгоритм взят из литературного источника, то его разработка и использование легче и надежнее. Итак, хороший алгоритм — необходимое, но не достаточное условие хорошей программы.

Индивидуальное задание

1. Составить схему алгоритма решения следующей задачи. Необходимо осуществить ввод последовательности положительных целых чисел до тех пор, пока не будет введено значение, равное 0. При вводе значения, отличного от целого числа, должно выдаваться предупреждение. Вывести количество введенных чисел, минимальное и максимальное из введенных значений и целое число, максимально близкое к среднему значению введенных чисел.

2. Составить схему алгоритма решения следующей задачи. Дана последовательность из N натуральных чисел. Необходимо за минимальное количество тактов переставить числа в последовательности таким образом, чтобы вначале оказались все нечетные числа, расположенные по возрастанию, а

в конце– четные, по убыванию. Программа должна обеспечивать ввод исходных данных, вывод результатов и диагностических сообщений в особых случаях.

3. Составить схему алгоритма решения следующей задачи. Для многочлена степени N необходимо ввести коэффициенты $a_0 \ldots a_N$ и вычислить значения полинома, используя схему Горнера для значений переменной x_i, последовательно вводимых с клавиатуры до тех пор, пока не будет введено значение 0.

4. Составить схему алгоритма, выполняющего вычисление биномиального коэффициента для значений n и m, вводимых с клавиатуры. Должна быть обеспечена работоспособность алгоритма при произвольном вводе пользователя.

5. Составить схему алгоритма решения системы двух линейных уравнений с двумя неизвестными любым способом с учетом всех вариантов решения. Программа должна обеспечивать ввод исходных данных, вывод результатов и диагностических сообщений в особых случаях.

6. Составить блок-схему алгоритма программы, выполняющей проверку: является ли произвольное положительное целое число простым. Программа должна обеспечивать ввод исходных данных, вывод результатов и диагностических сообщений в особых случаях.

7. Составить схему алгоритма, выполняющего вывод на экран всех простых чисел, меньше заданного. Необходимо предусмотреть обработку ошибок ввода и вывод диагностических сообщений для всех исключительных ситуаций.

8. Составить схему алгоритма, отвечающего на вопрос, являются ли два произвольных натуральных числа взаимно простыми. Необходимо обеспечить ввод исходных данных, вывод результатов и диагностических сообщений в особых случаях.

9. Составить схему алгоритма программы, выполняющей разложение произвольного положительного целого числа на элементарные множители. Программа должна обеспечивать ввод исходных данных, вывод результатов и диагностических сообщений в особых случаях.

10. Составить схему алгоритма программы, проверяющей, является ли произвольное целое число целой положительной степенью простого числа.

Программа должна обеспечивать ввод исходных данных, вывод результатов и диагностических сообщений в особых случаях.

11. Составить схему алгоритма программы, вычисляющей наибольшее простое число, являющееся общим делителем двух заданных натуральных чисел. Программа должна обеспечивать ввод исходных данных, вывод результатов и диагностических сообщений в особых случаях.

12. Составить схему алгоритма программы, вычисляющей наибольший общий делитель двух заданных положительных целых чисел. Программа должна обеспечивать ввод исходных данных, вывод результатов и диагностических сообщений в особых случаях.

13. Составить схему алгоритма программы, вычисляющей наименьшее общее кратное двух заданных положительных целых чисел. Программа должна обеспечивать ввод исходных данных, вывод результатов и диагностических сообщений в особых случаях.

14. Составить схему алгоритма, обеспечивающего проверку того, что три пары вещественных чисел, рассматриваемые как длины отрезков на плоскости, могут образовать равносторонний треугольник. Программа должна обеспечивать ввод исходных данных, вывод результатов и диагностических сообщений в особых случаях.

15. Составить схему алгоритма программы проверки того, что три пары чисел, представляющих декартовые координаты точек на плоскости, являются вершинами некоторого равнобедренного треугольника. Программа должна обеспечивать ввод исходных данных, вывод результатов и диагностических сообщений в особых случаях.

16. Составить схему алгоритма программы, определяющей, пересекаются ли два отрезка на плоскости, заданные декартовыми координатами своих концов. Программа должна обеспечивать ввод исходных данных, вывод результатов и диагностических сообщений в особых случаях.

17. Составить схему алгоритма программы проверки того, что три пары чисел, представляющих декартовые координаты точек на плоскости, являются вершинами некоторого прямоугольного треугольника. Программа должна обеспечивать ввод исходных данных, вывод результатов и диагностических сообщений в особых случаях.

18. Составить схему алгоритма программы, осуществляющей посимвольный ввод некоторого русского текста до ввода символа “конец файла”. Необходимо вывести общее количество введенных символов. Отношение в нем гласных букв к согласным и средней длины слова. Программа должна обеспечивать ввод исходных данных. Вывод результатов и диагностических сообщений в особых случаях.

19. Составить схему алгоритма программы, проверяющей выполнение неравенства $\frac{1}{n+1} < \ln\frac{n+1}{n} < \frac{1}{n}$ для произвольного целого положительного n. Программа должна обеспечивать ввод исходных данных. Вывод результатов и диагностических сообщений в особых случаях.

20. Составить схему алгоритма решения следующей задачи. Необходимо рассчитать значение функции $f(x) = 1+\sum_{k=1}^{\infty}(-1)^k \frac{x^{2k}}{(2k)!}$ в точке x.

21. Составить блок-схему алгоритма для вычисления квадратного корня x из вещественного числа y.

Примечание. Вычисление квадратного корня можно осуществить методом последовательного приближения с использованием зависимости $x_i = (x_{i-1}/y - y/x_{i-1})$до тех пор пока $(x_i * x_i) - y < 0.0001*y$, а x_0 выбирается между $y/4$ и $y/2$.

22. Составить схему алгоритма решения следующей задачи. Вводится N (N>5) пар чисел, представляющих собой декартовые координаты точек на плоскости. Необходимо построить окружность минимального радиуса, то есть определить координаты ее центра и радиус, которая охватывает все введенные точки. Необходимо предусмотреть обработку ошибок ввода и вывод диагностических сообщений для всех исключительных ситуаций.

23. Составить схему алгоритма решения следующей задачи. Вводится N (N>5) пар чисел, представляющих собой декартовые координаты точек на плоскости. Необходимо отобрать из них такое подмножество точек, соединение которых замкнутой ломаной линией образует выпуклый многоугольник, охватывающий все остальные точки. Необходимо предусмотреть обработку ошибок ввода и вывод диагностических сообщений для всех исключительных ситуаций.

24. Составить схему алгоритма решения следующей задачи. Имеется квадратная матрица целых чисел N*N, где N – нечетное число > 3. Необходимо за минимальное количество тактов переставить числа в матрице так, чтобы в центре находилось максимальное из них, а каждый из окружающих его квадратов содержал числа не меньшие, чем во внешнем по отношению к нему квадрате.

25. Составить схему алгоритма решения следующей задачи. Имеется квадратная матрица целых чисел N*N, где N – нечетное число > 3. Необходимо за минимальное количество тактов переставить числа в матрице так, чтобы в центре находилось минимальное из них, а каждый из окружающих его квадратов содержал числа не большие, чем во внешнем по отношению к нему квадрате.

26. Составить схему алгоритма решения следующей задачи. Дана квадратная матрица целых чисел размером N на N элементов. Необходимо осуществить обход элементов матрицы по спирали, начиная с верхнего левого угла по часовой стрелке. При обходе матрицы необходимо посчитать сумму элементов, отвечающих следующему условию: элемент является положительным нечетным числом и его величина больше, чем у элемента рассматриваемого перед ним.

27. Составить схему алгоритма решения следующей задачи. Дана матрица размером N*N квадратов, каждый из которых раскрашен в произвольном порядке одним из следующих цветов: красным, синим, желтым, зеленым. Необходимо за минимальное количество тактов переставить квадраты в матрице таким образом, чтобы квадрат каждого цвета хотя бы одной гранью соприкасался с квадратом того же цвета, а в углах матрицы находились, начиная с верхнего левого угла, квадраты перечисленных цветов. Программа должна обеспечивать ввод исходных данных, вывод результатов и диагностических сообщений в особых случаях.

28. Составить схему алгоритма решения следующей задачи. Необходимо осуществить ввод последовательности положительных целых чисел до тех пор, пока не будет введено значение, равное 0. При вводе значения, отличного от целого числа, должно выдаваться предупреждение. Вывести количество чисел в последовательности, среднее геометрическое и среднее арифметическое значение введенных чисел.

ЛАБОРАТОРНАЯ РАБОТА № 2

Тема : алгоритмические модели.

Цель: исследовать свойства алгоритмов на примере алгоритмической модели- машины Поста.

Краткие теоретические сведения

1.5. Устройство машины Поста

Прежде всего надо сказать, что машина Поста не есть реально существующее, сделанное кем-то устройство. Машина Поста, как и ее близкий родственник машина Тьюринга, представляет собой мысленную конструкцию, хотя устройство, позволяющее моделировать работу машины Поста в случае небольших программ и небольших объемов вычислений, было изготовлено в 1970 году в Симферопольском государственном университете. Однако для нас не будет существенным факт, что машины Поста на самом деле нет. Напротив, мы будем предполагать ее как бы "существующей".

Машина Поста состоит из ленты и каретки (называемой также считывающей и записывающей головкой). Лента бесконечна и разделена на секции одинакового размера. Порядок, в котором расположены секции ленты, подобен порядку, в котором расположены все целые числа. Поэтому естественно ввести на ленте "целочисленную систему координат", занумеровав секции числами ..., -3 , -2 , -1 , 0 , 1 , 2 , 3,...

В каждой секции ленты может быть либо ничего не записано (такая секция называется пустой), либо стоять метка "**V**" (тогда секция называется отмеченной).

Информация о том, какие секции пустые, а какие отмеченные, образует состояние ленты. Иными словами, состояние ленты - это распределение меток по ее секциям. На точном математическом языке состояние ленты - это функция, которая каждому числу (номеру секции) ставит в соответствие либо метку, либо "пусто". Состояние ленты, в процессе работы, может меняться.

Каретка может передвигаться вдоль ленты влево и вправо. Когда она неподвижна, она стоит против ровно одной секции ленты; говорят, что каретка обозревает эту секцию, или держит ее в поле зрения. Информация о том, какие

секции пустые, а какие отмеченные и где стоит каретка, образуют состояние машины Поста.

Таким образом, состояние машины Поста слагается из состояния ленты и указания номера той секции, которую обозревает каретка.

За единицу времени, которая называется шагом, каретка может сдвинуться на одну секцию влево или вправо. Кроме того, каретка может поставить (напечатать) или уничтожить (стереть) метку в той секции, против которой она стоит, а также распознать, стоит или нет метка в обозреваемой ею секции.

Работа машины Поста состоит в том, что каретка передвигается вдоль ленты и печатает или стирает метки. Эта работа происходит по инструкции определенного вида, называемого программой. Для машины Поста возможно составление различных программ из разного числа команд, но эти команды строго определены.

1.6. Работа машины Поста

Работа машины Поста состоит в том, что каретка передвигается вдоль ленты и печатает или стирает метки. Эта работа происходит по инструкции определенного вида, называемого программой. Для машины Поста возможно составление различных программ из набора команд, но эти команды строго определены. Командой машины Поста называется выражение, имеющее один из следующих шести видов:

1) команда движения вправо: m . => n
2) команда движения влево: m . <= n
3) команда печати метки: m. p n
4) команда стирания метки: m. s n
5) команда передачи: m. ? n1,n2
6) команда остановки: m. stop

Число m, стоящее в начале команды, называется номером команды, а число n, стоящее после команды (а у команды передачи управления n1и n2), называется отсылкой. У команды остановки отсылка отсутствует.

Программой машины Поста называется конечный непустой (т.е. содержащий хотя бы одну команду) список команд машины Поста, обладающий следующими двумя свойствами.

1. На первом месте в этом списке стоит команда с номером 1, на втором месте (если оно есть) - команда с номером 2 и т.д.; вообще на k-м месте стоит команда с номером k.

2. Отсылка любой из команд списка совпадает с номером некоторой (другой или той же самой) команды списка (более точно: для каждой отсылки каждой команды списка найдется в списке такая команда, номер которой равен рассматриваемой отсылке).

Чтобы машина Поста работала, надо задать некоторую программу и некоторое состояние машины, т.е. как-то расставить метки по секциям ленты (в частности, можно все секции оставить пустыми) и поставить каретку против одной из секций.

Работа машины на основании заданной программы происходит следующим образом. Машина приводится в начальное состояние и приступает к выполнению первой команды программы. Каждая команда выполняется за один шаг, а переход от выполнения одной команды к выполнению другой происходит по следующему правилу: пусть на k-м шаге выполнялась команда с номером m, тогда,

1) если эта команда имеет единственную отсылку n, то на (k+1)-м шаге выполняется команда с номером n;

2) если эта команда имеет две отсылки n1 и n2, то на (k+1)-м шаге выполняется одна из двух команд - с номером n1 или с номером n2;

3) если же выполняющаяся на k-м шаге команда вовсе не имеет отсылки, то на (k+1)-м шаге и на всех последующих шагах не выполняется никакая команда - машина останавливается.

Выполнение команды движения вправо состоит в том, что каретка сдвигается на одну секцию вправо.

Выполнение команды движения влево состоит в том, что каретка сдвигается на одну секцию влево.

Выполнение команды печати метки состоит в том, что каретка ставит метку на обозреваемой секции. Выполнение этой команды возможно лишь в том случае, если обозреваемая перед началом выполнения команды секция пуста.

Если же на обозреваемой секции уже стоит метка, то команда считается невыполнимой.

Выполнение команды стирания метки состоит в том, что каретка уничтожает (стирает) метку в обозреваемой секции. Выполнение этой команды возможно лишь в том случае, если обозреваемая секция отмечена (в ней стоит метка). Если же на обозреваемой секции нет метки, команда считается невыполнимой.

Выполнение команды передачи управления с отсылками n1 и n2 никак не изменяет состояние машины: ни одна из меток не уничтожается и не ставится, и каретка также остается неподвижной (машина делает "шаг на месте"). Однако если секция, обозреваемая перед началом выполнения команды, была отмечена, то следующей должна выполняться команда с номером n1. Если же эта секция была пустая, то следующая должна выполняться команда с номером n2 (роль команды передачи управления сводится, следовательно, к тому, что каретка во время выполнения этой команды как бы "распознает", стоит ли перед ней метка).

Выполнение команды остановки тоже никак не меняет состояния машины и состоит в том, что машина останавливается.

Если теперь, задав программу и какое-либо начальное состояние, пустить машину в ход, то осуществится один из следующих трех вариантов.

1. В ходе выполнения программы машина дойдет до выполнения невыполнимой команды (печать метки в непустой секции или стирание метки уже в пустом месте). Выполнение программы тогда прекращается, машина останавливается- происходит безрезультативная остановка (аварийная остановка).

2. В ходе выполнения программы машина дойдет до выполнения команды остановки. Программа в этом случае считается выполненной, машина останавливается - происходит результативная остановка (нормальное завершение).

3. В ходе выполнения программы машина не дойдет до выполнения ни одной из команд, указанных в первых двух вариантах. Выполнение программы при этом никогда не прекращается, машина никогда не останавливается - процесс работы машины происходит бесконечно (зацикливание).

При зацикливании и аварийном завершении не получаем никаких результатов. При нормальном завершении получаем результаты, которые могут быть правильными или неправильными. Правильные результаты возможны при правильно составленном алгоритме и соответствующих входных данных.

В качестве примера составим программу для следующей задачи. Необходимо сложить два натуральных числа, находящихся на произвольном расстоянии. Начальное положение головки- где-то над первым числом.

1.	<=	2	Поиск начала первого числа: сдвигаемся влево
2.	?	3,1	до тех пор, пока не встретим не отмеченную ячейку.
3.	=>	4	Сдвинуться вправо на одну метку и
4.	s	5	и удалить ее.
5.	=>	6	Поиск конца первого числа: сдвиг вправо
6.	?	7,5	пока головка не встанет на неотмеченную ячейку и
7.	p	8	ставим ее
8.	=>	9	Проверить заполнен ли промежуток между числами и,
9.	?	1,10	если промежуток не заполнился, то переход на начало.
10.	stop		Конец программы

Индивидуальное задание

1. Записать программу для машины Поста, выполняющую сложение двух натуральных чисел, записанных на произвольном расстоянии друг от друга. Начальное положение головки – на произвольном расстоянии слева от первого числа.

2. Составить программу для машины Поста, выполняющую сложение двух натуральных чисел, записанных на произвольном расстоянии друг от друга. Начальное положение головки – на произвольном расстоянии справа от второго числа.

3. Составить программу для машины Поста, выполняющую сложение двух натуральных чисел, записанных на произвольном расстоянии друг от друга. Начальное положение головки – в произвольном месте между числами.

4. Составить программу для машины Поста, выполняющую перестановку двух натуральных чисел, записанных на произвольном расстоянии друг от друга. Начальное положение головки – на произвольном расстоянии справа от второго числа.

5. Составить программу для машины Поста, выполняющую перестановку двух натуральных чисел, записанных на произвольном расстоянии друг от друга. Начальное положение головки – на произвольном расстоянии слева от первого числа.

6. Составить программу для машины Поста, выполняющую перестановку двух натуральных чисел, записанных на произвольном расстоянии друг от друга. Начальное положение головки – в произвольном месте между числами.

7. Составить программу для машины Поста для решения следующей задачи. На ленте под головкой записано некоторое натуральное число. После выполнения программы головка должна быть установлена справа от числа, если оно нечетное, или слева от него – если оно четное.

8. Составить программу для машины Поста для решения следующей задачи. На ленте под головкой записано некоторое натуральное число. После выполнения программы головка должна быть установлена слева от числа, если оно нечетное, или справа от него – если оно четное.

9. Записать программу для машины Поста, выполняющую умножение двух натуральных чисел, записанных на произвольном расстоянии друг от друга. Начальное положение головки – на произвольном расстоянии слева от первого числа.

10. Составить программу для машины Поста, выполняющую умножение двух натуральных чисел, записанных на произвольном расстоянии друг от друга. Начальное положение головки – на произвольном расстоянии справа от второго числа.

11. Составить программу для машины Поста, выполняющую умножение двух натуральных чисел, записанных на произвольном расстоянии друг от друга. Начальное положение головки – в произвольном месте между числами.

12. Составить программу для машины Поста целочисленного деления чисел на 2.

13. Составить программу для машины Поста целочисленного деления чисел на 3.

14. Составить программу для машины Поста целочисленного деления чисел на произвольное число k.

15. Составить программу для машины Поста умножения двух чисел.

16. Составить программу для машины Поста возведения чисел в квадрат.

17. Составить программу для машины Поста деления одного числа на другое.

РАЗДЕЛ 2. ОСНОВЫ ПРОГРАММИРОВАНИЯ НА ЯЗЫКЕ СИ

2.1. Элементы Языка СИ

2.1.1. Используемые символы

Множество символов, используемых в языке СИ, можно разделить на пять групп.

1. Символы, используемые для образования ключевых слов и идентификаторов. В эту группу входят прописные и строчные буквы английского алфавита, а также символ подчеркивания. Следует отметить, что одинаковые прописные и строчные буквы считаются различными символами, так как имеют различные коды.

2. Группа прописных и строчных букв русского алфавита и арабские цифры

3. Знаки нумерации и специальные символы:

, . ; : ? ' ! | /\ ~ * + - ( ) { } < > [ ] # % & ^ = "

Эти символы используются с одной стороны для организации процесса вычислений, а с другой - для передачи компилятору определенного набора инструкций.

4. Управляющие и разделительные символы. К той группе символов относятся: пробел, символы табуляции, перевода строки, возврата каретки, новая страница и новая строка. Эти символы отделяют друг от друга объекты, определяемые пользователем, к которым относятся константы и идентификаторы. Последовательность разделительных символов рассматривается компилятором как один символ (последовательность пробелов).

5. Кроме выделенных групп символов, в языке СИ широко используются так называемые, управляющие последовательности, т.е. специальные символьные комбинации, используемые в функциях ввода и вывода информации. Управляющая последовательность строится на основе использования обратной дробной черты (\) (обязательный первый символ) и комбинацией латинских букв и цифр:

\a –звонок; \b - возврат на шаг; \t - горизонтальная табуляция; \n-переход на новую строку; \v- вертикальная табуляция; \r - возврат каретки; \f - перевод формата; \" – кавычки; \' апостроф; \0 - ноль-символ; \\ - обратная дробная черта; \ddd - символ набора кодов ПЭВМ в восьмеричном представлении; \xddd - символ набора кодов ПЭВМ в шестнадцатеричном представлении.

Последовательности вида \ddd и \xddd (здесь d обозначает цифру) позволяет представить символ из набора кодов ПЭВМ как последовательность восьмеричных или шестнадцатеричных цифр соответственно. Например, символ возврата каретки может быть представлен различными способами:

\r - общая управляющая последовательность,

\015 - восьмеричная управляющая последовательность,

\x00D - шестнадцатеричная управляющая последовательность.

2.1.2. Константы

Константами называются перечисление величин в программе. В языке СИ разделяют четыре типа констант: целые константы, константы с плавающей запятой, символьные константы и строковые литералы.

Целая константа: это десятичное, восьмеричное или шестнадцатеричное число, которое представляет целую величину в одной из следующих форм: десятичной, восьмеричной или шестнадцатеричной.

Десятичная константа состоит из одной или нескольких десятичных цифр, причем первая цифра не должна быть нулем (в противном случае число будет воспринято как восьмеричное).

Восьмеричная константа состоит из обязательного нуля и одной или нескольких восьмеричных цифр (среди цифр должны отсутствовать восьмерка и девятка, так как эти цифры не входят в восьмеричную систему счисления).

Шестнадцатеричная константа начинается с обязательной последовательности 0x или 0X и содержит одну или несколько шестнадцатеричных цифр (цифры, представляющие собой набор цифр, шестнадцатеричной системы счисления: 0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F)

Примеры целых констант:

Десятичная константа	Восьмеричная константа	Шестнадцатеричная константа
16	020	0x10
127	0117	0x2B
240	0360	0XF0

Если требуется сформировать отрицательную целую константу, то используют знак "-" перед записью константы (который будет называться унарным минусом). Например: -0x10, -020, -16 .

Каждой целой константе присваивается тип, определяющий преобразования, которые должны быть выполнены, если константа используется в выражениях. Тип константы определяется следующим образом:

- десятичные константы рассматриваются как величины со знаком и им присваивается тип int (целая) или long (длинная целая) в соответствии со значением константы. Если константа меньше 32768, то ей присваивается тип int, в противном случае - long.

- восьмеричным и шестнадцатеричным константам присваивается тип int, unsigned int (беззнаковая целая), long или unsigned long в зависимости от значения константы согласно табл. 2.1.

Таблица 2.1

Диапазон констант

Диапазон шестнадцатеричных констант	Диапазон восьмеричных констант	Тип
0x0 – 0x7FFF	0 – 077777	int
0X8000 – 0XFFFF	0100000 – 0177777	unsigned int
0X10000 - 0X7FFFFFFF	0200000 – 017777777777	long
0X80000000 - 0XFFFFFFFF	020000000000-037777777777	unsigned long

Для того чтобы любую целую константу определить типом long, достаточно в конце константы поставить букву "l" или "L". Пример:

5l, 6l, 128L, 0105L, OX2A11L.

Константа с плавающей точкой - десятичное число, представленное в виде действительной величины с десятичной точкой или экспонентой. Формат имеет вид

[цифры].[цифры] [E|e [+|-] цифры]

Число с плавающей точкой состоит из целой и дробные части и (или) экспоненты. Константы с плавающей точкой представляют положительные величины удвоенной точности (имеют тип double). Для определения отрицательной величины необходимо сформировать константное выражение, состоящее из знака минуса и положительной константы.

Примеры: 115.75, 1.5E-2, -0.025, .075, -0.85E2

Символьная константа - представляется символом, заключенном в апострофы. Управляющая последовательность рассматривается как одиночный символ, допустимо ее использовать в символьных константах. Значением символьной константы является числовой код символа. Примеры:

' '- пробел ,

'Q'- буква Q ,

'\n' - символ новой строки ,

'\\' - обратная дробная черта ,

'\v' - вертикальная табуляция .

Символьные константы имеют тип int и при преобразовании типов дополняются знаком.

Строковая константа (литерал) - последовательность символов, включая строковые и прописные буквы русского и латинского, а также цифры заключенные в кавычки (") .

Например: "БГУИР каф. ИИТ", "город Минск", "YZPT КОД".

Отметим, что все управляющие символы, кавычка ("), обратная дробная черта (\) и символ новой строки в строковом литерале и в символьной константе представляются соответствующими управляющими последовательностями. Каждая управляющая последовательность представляется как один символ. Например, при печати литерала "БГУИР \n каф. ИИТ" его часть "БГУИР" будет напечатана на одной строке, а вторая часть "каф. ИИТ" - на следующей строке.

Символы строкового литерала сохраняются в области оперативной памяти. **В конец каждого строкового литерала компилятором добавляется нулевой символ, представляемый управляющей последовательностью \0.**

Строковый литерал имеет тип char[] . Это означает, что **строка рассматривается как массив символов**. Отметим важную особенность, число элементов массива равно числу символов в строке плюс 1, так как нулевой символ (символ конца строки) также является элементом массива. Все

строковые литералы рассматриваются компилятором как различные объекты. Строковые литералы могут располагаться на нескольких строках. Такие литералы формируются на основе использования обратной дробной черты и клавиши ввод. Для сцепления строковых литералов можно использовать символ (или символы) пробела. Если в программе встречаются два или более строковых литерала, разделенные только пробелами, то они будут рассматриваться как одна символьная строка. Этот принцип можно использовать для формирования строковых литералов, занимающих более одной строки.

2.1.3. Идентификатор

Идентификатором называется последовательность цифр и букв, а также специальных символов при условии, что первой стоит буква или специальный символ. Для образования идентификаторов могут быть использованы строчные или прописные буквы латинского алфавита. В качестве специального символа может использоваться символ подчеркивание (_). Два идентификатора, для образования которых используются совпадающие строчные и прописные буквы, считаются различными. Например: abc, ABC, A128B, a128b .

Важной особенностью является то, что компилятор допускает любое количество символов в идентификаторе, хотя значимыми являются первые 31 символ. Идентификатор создается на этапе объявления переменной, функции, структуры и т.п., после этого его можно использовать в последующих операторах разрабатываемой программы. Следует отметить важные особенности при выборе идентификатора.

Во-первых, идентификатор не должен совпадать с ключевыми словами, с зарезервированными словами и именами функций библиотеки компилятора языка СИ.

Во-вторых, следует обратить особое внимание на использование символа подчеркивание в качестве первого символа идентификатора, поскольку идентификаторы построенные таким образом, что, с одной стороны, могут совпадать с именами системных функций и (или) переменных, а с другой стороны, при использовании таких идентификаторов программы могут оказаться непереносимыми, т.е. их нельзя использовать на компьютерах других типов.

В третьих, на идентификаторы, используемые для определения внешних переменных, должны быть наложены ограничения, формируемые используемым редактором связей (отметим, что использование различных версий редактора связей или различных редакторов накладывает различные требования на имена внешних переменных).

2.1.4. Ключевые слова

Ключевые слова - это зарезервированные идентификаторы, которые наделены определенным смыслом. Их можно использовать только в соответствии со значением известным компилятору языка СИ.

Приведем список ключевых слов.

```
auto      double    int   struct  break   else   long   switch
register  tupedef   char  extern  return  void   case   float
unsigned  default   for   signed  union   do     if     sizeof
volatile  continue  enum  short   while
```

Кроме того, в рассматриваемой версии реализации языка СИ, зарезервированными словами являются :

_asm, fortran, near, far, cdecl, huge, paskal, interrupt .

Ключевые слова far, huge, near позволяют определить размеры указателей на области памяти. Ключевые слова _asm, cdelc, fortran, pascal служат для организации связи с функциями, написанными на других языках, а также для использования команд языка ассемблера непосредственно в теле разрабатываемой программы на языке СИ.

Ключевые слова не могут быть использованы в качестве идентификаторов.

2.1.5. Использование комментариев в тексте программы

Комментарий - это набор символов, которые игнорируются компилятором, на этот набор символов, однако, накладываются следующие ограничения. Внутри набора символов, который представляет комментарий, не может быть специальных символов, определяющих начало и конец комментариев соответственно (/* и */). Отметим, что комментарии могут заменить как одну строку, так и несколько. Например:

```
/* комментарии к программе */
     или
/* комментарии  можно  записать в следующем виде,  однако надо
```

```
быть осторожным, чтобы внутри последовательности, которая
игнорируется компилятором, не попались операторы программы,
которые также будут игнорироваться */
```

Неправильное определение комментариев.

```
/* комментарии к алгоритму /* решение краевой задачи */ */
```

или

```
/* комментарии к алгоритму решения */ краевой задачи */
```

В языке Си++ допускается использовать построчные комментарии. Символы // начинают комментарий, который заканчивается в конце строки, на которой они появились.

```
double a,b,c     // a,b,c - коэффициенты квадратного трехчлена
```

2.2. Типы данных и их объявление

В языке СИ необходимо объявлять все переменные, используемые в программе, явно вместе с указанием соответствующих им типов.

Объявления переменной имеет следующий формат:

```
[спецафикатор-класа-памяти] спецификатор-типа
  описатель [=инициатор] [,описатель [= инициатор] ]...
```

Описатель - идентификатор простой переменной, либо более сложная конструкция с квадратными скобками, круглыми скобками или звездочкой (набором звездочек).

Спецификатор типа - одно или несколько ключевых слов, определяющих тип объявляемой переменной. В языке СИ имеется стандартный набор типов данных, используя который можно сконструировать новые (уникальные) типы данных.

Инициатор - задает начальное значение или список начальных значений, которые (которое) присваивается переменной при объявлении.

Спецификатор класса памяти - определяется одним из четырех ключевых слов языка СИ: auto, extern, register, static, и указывает, каким образом будет распределяться память под объявляемую переменную, с одной стороны, а с другой, область видимости этой переменной, т.е., из каких частей программы можно к ней обратиться.

2.2.1 Категории типов данных

Ключевые слова для определения основных типов данных :

```
Целые типы :                    Плавающие типы:
```

```
char                    float
int                     double
short                   long  double
long
signed
unsigned
```

Переменная любого типа может быть объявлена как немодифицируемая. Это достигается добавлением ключевого слова const к спецификатору-типа. Объекты с типом const представляют собой данные, используемые только для чтения, т.е. этой переменной не может быть присвоено новое значение. Отметим, что если после слова const отсутствует спецификатор-типа, то подразумевается спецификатор типа int. Если ключевое слово const стоит перед объявлением составных типов (массив, структура, смесь, перечисление), то это приводит к тому, что каждый элемент также должен являться немодифицируемым, т.е. значение ему может быть присвоено только один раз.

Примеры:

```
const double A=2.128E-2;
const B=286; (подразумевается const int B=286)
```

2.2.2. Целый тип данных

Для определения данных целого типа используются различные ключевые слова, которые определяют диапазон значений и размер области памяти, выделяемой под переменные (табл. 2.2).

Таблица 2.2

Диапазон значений целых переменных.

Тип	Размер памяти в байтах	Диапазон значений
char	1	от -128 до 127
int	Для IBM XT,AT,SX,DX 2	
short	2	от -32768 до 32767
long	4	от -2 147 483 648 до 2 147 483 647
unsigned shar	1	от 0 до 255
unsigned int	Для IBM XT,AT,SX,DX 2	
unsigned short	2	от 0 до 65535
unsigned long	4	от 0 до 4 294 967 295

Отметим, что ключевые слова signed и unsigned необязательны. Они указывают, как интерпретируется нулевой бит объявляемой переменной, т.е. если указано ключевое слово unsigned, то нулевой бит интерпретируется как часть числа, в противном случае нулевой бит интерпретируется как знаковый. В случае отсутствия ключевого слова unsigned целая переменная считается знаковой. В том случае, если спецификатор типа состоит из ключевого типа signed или unsigned и далее следует идентификатор переменной, то она будет рассматриваться как переменная типа int. Например:

```
unsigned int n;
unsigned int b;
int c;       (подразумевается  signed   int c  );
unsigned d;  (подразумевается  unsigned int d  );
signed f;    (подразумевается  signed   int f  ).
```

Отметим, что модификатор-типа char используется для представления символа (из массива представление символов) или для объявления строковых литералов. Значением объекта типа char является код (размером 1 байт), соответствующий представляемому символу. Для представления символов русского алфавита модификатор типа идентификатора данных имеет вид unsigned char, так как коды русских букв превышают величину 127.

Следует сделать следующее замечание: в языке СИ не определено представление в памяти и диапазон значений для идентификаторов с модификаторами-типа int и unsigned int. **Размер памяти для переменной с модификатором типа int определяется длиной машинного слова, которое имеет различный размер на разных машинах.** Так, на 16-разрядных машинах размер слова равен 2-м байтам, на 32-разрядных машинах соответственно 4-м байтам, т.е. тип int эквивалентен типам short int, или long int в зависимости от архитектуры используемой ПЭВМ. Таким образом, одна и та же программа может правильно работать на одном компьютере и неправильно на другом. Для определения длины памяти, занимаемой переменной, можно использовать операцию sizeof (см. п.2.3.1) языка СИ, возвращающую значение длины указанного модификатора-типа.

Например:

```
a = sizeof(int);
b = sizeof(long int);
c = sizeof(unsigned long);
d = sizeof(short);
```

Отметим также, что восьмеричные и шестнадцатеричные константы также могут иметь модификатор unsigned. Это достигается указанием префикса u или U после константы, константа без этого префикса считается знаковой.

Например:

```
0xA8C   (int  signed   );
01786l  (long signed   );
0xF7u   (int  unsigned );
```

2.2.3. Данные плавающего типа

Для переменных, представляющих число с плавающей точкой, используются следующие модификаторы типа: float, double, long double (в некоторых реализациях языка long double СИ отсутствует).

Величина с модификатором-типа float занимает 4 байта. Из них 1 бит отводится для знака, 8 бит для избыточной экспоненты и 23 бита для мантиссы. Отметим, что старший бит мантиссы всегда равен 1, поэтому он не заполняется, в связи с этим диапазон значений переменной с плавающей точкой приблизительно равен от 3.14E-38 до 3.14E+38.

Величина типа double занимает 8 байт в памяти. Ее формат аналогичен формату float. Биты памяти распределяются следующим образом: 1 бит для знака, 11 бит для экспоненты и 52 бита для мантиссы. С учетом опущенного старшего бита мантиссы диапазон значений равен от 1.7E-308 до 1.7E+308.

Примеры:

```
float  f, a, b;
double x, y;
```

2.2.4. Указатели

Указатель - это адрес памяти, распределяемой для размещения идентификатора (в качестве идентификатора может выступать имя переменной, массива, структуры, строкового литерала). В случае если переменная объявлена как указатель, то она содержит адрес памяти, по которому может находиться скалярная величина любого типа. При объявлении переменной типа указатель необходимо определить тип объекта данных, адрес которых будет содержать переменная, и имя указателя с предшествующей звездочкой (или группой звездочек). Формат объявления указателя:

спецификатор-типа [модификатор] * описатель

Спецификатор-типа задает тип объекта и может быть любого основного типа, типа структуры, смеси (об этом будет сказано ниже). Задавая вместо спецификатора-типа ключевое слово void, можно своеобразным образом отсрочить спецификацию типа, на который ссылается указатель. Переменная, объявляемая как указатель на тип void, может быть использована для ссылки на объект любого типа. Однако для того чтобы можно было выполнить арифметические и логические операции над указателями или над объектами, на которые они указывают, необходимо при выполнении каждой операции явно определить тип объектов. Такие определения типов могут быть выполнены с помощью операции приведения типов.

В качестве модификаторов при объявлении указателя могут выступать ключевые слова const, near, far, huge. Ключевое слово const указывает, что указатель не может быть изменен в программе. Размер переменной, объявленной как указатель, зависит от архитектуры компьютера и от используемой модели памяти, для которой будет компилироваться программа. Указатели на различные типы данных не обязательно должны иметь одинаковую длину.

Для модификации размера указателя можно использовать ключевые слова near, far, huge.

Примеры:

```
unsigned int * a; /* переменная a представляет собой
      указатель на тип unsigned int (целые числа без знака) */
double * x;       /* переменная x указывает на тип данных
      с плавающей точкой удвоенной точности                 */
char * fuffer ;   /* объявляется указатель с именем fuffer,
      который указывает на переменную типа char             */
double nomer;

void *addres;
addres = & nomer;
(double *)addres ++;
```

/* Переменная addres объявлена как указатель на объект любого типа. Поэтому ей можно присвоить адрес любого объекта (& - операция вычисления адреса). Однако, ни одна арифметическая операция не может быть выполнена над указателем, пока не будет явно определен тип данных, на которые он указывает. Это можно сделать, используя операцию приведения типа (double *) для преобразования addres к указателю на тип double, а затем увеличение адреса. */

```
const * dr;
```

/* Переменная dr объявлена как указатель на константное выражение, т.е. значение указателя может изменяться в процессе выполнения программы, а величина, на которую он указывает, нет. */

```
unsigned char * const w = &obj.
```

/* Переменная w объявлена как константный указатель на данные типа char unsigned. Это означает, что на протяжение всей программы w будет указывать на одну и ту же область памяти. Содержание же этой области может быть изменено. */

2.2.5. Переменные перечислимого типа

Переменная, которая может принимать значение из некоторого списка значений, называется переменной перечислимого типа, или перечислением.

Объявление перечисления начинается с ключевого слова enum и имеет два формата представления.

Формат 1. enum [имя-тега-перечисления] {список-перечисления}
описатель[,описатель...];

Формат 2. enum имя-тега-перечисления описатель [,описатель..];

Объявление перечисления задает тип переменной перечисления и определяет список именованных констант, называемый список-перечисление. Значением каждого имени списка является некоторое целое число.

Переменная типа перечисления может принимать значения одной из именованных констант списка. Именованные константы списка имеют тип int. Таким образом, память соответствующая переменной перечисления, это память, необходимая для размещения значения типа int.

Переменная типа enum могут использоваться в индексных выражениях и как операнды в арифметических операциях и в операциях отношения.

В первом формате 1 имена и значения перечисления задаются в списке перечислений. Необязательное имя-тега-перечисления - это идентификатор, который именует тег перечисления, определенный списком перечисления. Описатель именует переменную перечисления. В объявлении может быть задана более чем одна переменная типа перечисления.

Список-перечисления содержит одну или несколько конструкций вида
идентификатор [= константное выражение]

Каждый идентификатор именует элемент перечисления. Все идентификаторы в списке enum должны быть уникальными. В случае отсутствия константного выражения первому идентификатору соответствует значение 0, следующему идентификатору - значение 1 и т.д. Имя константы перечисления эквивалентно ее значению.

Идентификатор, связанный с константным выражением, принимает значение, задаваемое этим константным выражением. Константное выражение должно иметь тип int и может быть как положительным, так и отрицательным. Следующему идентификатору в списке присваивается значение, равное константному выражению плюс 1, если этот идентификатор не имеет своего константного выражения. Использование элементов перечисления должно подчиняться следующим правилам:

1. Переменная может содержать повторяющиеся значения.

2. Идентификаторы в списке перечисления должны быть отличны от всех других идентификаторов в той же области видимости, включая имена обычных переменных и идентификаторы из других списков перечислений.

3. Имена типов перечислений должны быть отличны от других имен типов перечислений, структур и смесей в этой же области видимости.

4. Значение может следовать за последним элементом списка перечисления.

Пример:

```
enum week { SUB = 0,     /*    0   */
            VOS = 0,     /*    0   */
            POND,        /*    1   */
            VTOR,        /*    2   */
            SRED,        /*    3   */
            HETV,        /*    4   */
            PJAT         /*    5   */
            }   rab_ned ;
```

В данном примере объявлен перечислимый тег week с соответствующим множеством значений и объявлена переменная rab_ned имеющая тип week.

Во втором формате используется имя тега перечисления для ссылки на тип перечисления, определяемый где-то в другом месте. Имя тега перечисления должно относиться к уже определенному тегу перечисления в пределах текущей области видимости. Так как тег перечисления объявлен где-то в другом месте, список перечисления не представлен в объявлении.

Пример:

```
enum week rab1;
```

В объявлении указателя на тип данных перечисления и объявляемых typedef для типов перечисления можно использовать имя тега перечисления до того, как данный тег перечисления определен. Однако определение перечисления должно предшествовать любому действию используемого указателя на тип объявления typedef. Объявление без последующего списка описателей описывает тег, или, если так можно сказать, шаблон перечисления.

2.3. Выражения и присваивания

2.3.1. Операнды и операции

Комбинация знаков операций и операндов, результатом которой является определенное значение, называется выражением. Знаки операций определяют действия, которые должны быть выполнены над операндами. Каждый операнд в выражении может быть выражением. Значение выражения зависит от расположения знаков операций и круглых скобок в выражении, а также от приоритета выполнения операций.

В языке СИ присваивание также является выражением, и значением такого выражения является величина, которая присваивается.

При вычислении выражений тип каждого операнда может быть преобразован к другому типу. Преобразования типов могут быть неявными, при выполнении операций и вызовов функций, или явными, при выполнении операций приведения типов.

Операнд - это константа, литерал, идентификатор, вызов функции, индексное выражение, выражение выбора элемента или более сложное выражение, сформированное комбинацией операндов, знаков операций и круглых скобок. Любой операнд, который имеет константное значение, называется константным выражением. Каждый операнд имеет тип.

Операции. По количеству операндов, участвующих в операции, операции подразделяются на унарные, бинарные и тернарные.

В языке Си имеются следующие унарные операции:

-	арифметическое отрицание (отрицание и дополнение);
~	побитовое логическое отрицание (дополнение);
!	логическое отрицание;
*	разадресация (косвенная адресация);
&	вычисление адреса;
+	унарный плюс;
++	увеличение (инкремент);
--	уменьшение (декремент);
sizeof	размер .

Унарные операции выполняются справа налево.

Операции увеличения и уменьшения увеличивают или уменьшают значение операнда на единицу и могут быть записаны как справа, так и слева от операнда. Если знак операции записан перед операндом (префиксная форма), то изменение операнда происходит до его использования в выражении. Если знак операции записан после операнда (постфиксная форма), то операнд вначале используется в выражении, а затем происходит его изменение.

В отличие от унарных бинарные операции, список которых приведен в табл.2.3, выполняются слева направо.

Таблица 2.3

Бинарные операции

Знак операции	Операция	Пример
1	2	3
Арифметические операции		
*	Умножение	a = b * c; если b=2, c=3, то a=6
/	Деление	a = b / c; если b=5.0,c=2.0, то a=2.5
%	Остаток от деления (деление по модулю)	a = b % c; если b=5,c=2, то a=1 (операция используется только для переменных целого типа)
+	Сложение	a = b + c; если b=2,c=3, то a=5
-	Вычитание	a = b - c; если b=2,c=3, то a=-1
Операции сдвига		
<<	Сдвиг влево	a = b << c; осуществляется сдвиг значения b влево на c разрядов; в освободившееся разряды b заносятся нули. Если $b=9=1001_2$, c=2, то $a=36=100100_2$ Сдвиг влево соответствует умножению первого операнда на степень двойки

Продолжение табл. 2.3

1	2	3
>>	Сдвиг вправо	a = b >> c; осуществляется сдвиг значения b вправо на c разрядов; в освободившееся разряды b заносятся нули для положительных b и заполняются копией знакового бита для отрицательных b. Если $b=9=1001_2$, c=2, то $a=2=0010_2$ Сдвиг влево соответствует делению первого операнда на степень двойки
Операции отношения		
<	Меньше	a<b; дает результат 1, если a меньше b, и 0 – в противном случае
<=	Меньше или равно	a<=b; дает результат 1, если a меньше или равно b, и 0 – в противном случае
>	Больше	a>b; дает результат 1, если a больше b, и 0 – в противном случае
>=	Больше или равно	a>=b; дает результат 1, если a больше или равно b, и 0 – в противном случае
= =	Равно	a = = b; дает результат 1, если a равно b, и 0 – в противном случае
!=	Не равно	a!=b; дает результат 1, если a не равно b, и 0, если a равно b
Поразрядные операции		
&	Поразрядное И	a = b & c; если $b=10=1010_2$, $c=6=0110_2$, то $a=2=0010_2$
|	Поразрядное ИЛИ	a = b | c; если $b=10=1010_2$, $c=6=0110_2$, то $a=14=1110_2$
^	Поразрядное исключающее ИЛИ	a = b ^ c; если $b=12=1100_2$, $c=6=0110_2$, то $a=10=1010_2$

Окончание табл. 2.3

1	2	3
Логические операции		
&&	Логическое И	a = b && c; если b и c не равны нулю, то a=1, в противном случае a=0
||	Логическое ИЛИ	a = b || c; если b и c равны нулю, то a=0, в противном случае a=1
Последовательного вычисления		
,	Последовательное вычисление	i =0; j=0; эквивалентно i=0,j=0;
Операции присваивания		
=	Присваивание	a=0;
*=	Умножение с присваиванием	a*=b; если a=2,b=5, то результат a=10
/=	Деление с присваиванием	a%=b; если a=5.0,b=2.0, то результат a=2.5
%=	Остаток от деления с присваиванием	a%=b; если a=5,b=3, то результат a=2
-=	Вычитание с присваиванием	a-=b; если a=2,b=5, то результат a=-3
+=	Сложение с присваиванием	a+=b; если a=2,b=5, то результат a=7
<<=	Сдвиг влево с присваиванием	a<<=1; если a=9,b=2, то результат a=36
>>=	Сдвиг вправо с присваиванием	a>>=b; если a=8,b=1, то результат a=4
&=	Поразрядное И с присваиванием	a&=b; если a=10,b=6, то результат a=2
|=	Поразрядное ИЛИ с присваиванием	a|=b; если a=10,b=6, то результат a=14
^=	Поразрядное исключающее ИЛИ с присваиванием	a^=b; если a=12,b=6, то результат a=10

Левый операнд операции присваивания должен быть выражением, ссылающимся на область памяти (но не объектом, объявленным с ключевым словом const), такие выражения называются леводопустимыми.

При записи выражений следует помнить, что символы (*), (&), (!), (+) могут обозначать унарную или бинарную операцию.

2.3.2. Приоритеты операций и порядок вычислений

В языке СИ операции с высшими приоритетами вычисляются первыми. Наивысшим приоритетом является приоритет, равный 1. Приоритеты и порядок операций приведены в табл. 2.4.

Таблица 2.4

Приоритет операций

Приоритет	Знак операции	Типы операции	Порядок выполнения
1	() [] . ->	Выражение	Слева направо
2	- ~ ! * & ++ -- sizeof приведение типов	Унарные	Справа налево
3	* / %	Мультипликативные	Слева направо
4	+ -	Аддитивные	
5	<< >>	Сдвиг	
6	< > <= >=	Отношение	
7	== !=	Отношение (равенство)	
8	&	Поразрядное И	
9	^	Поразрядное исключающее ИЛИ	
10	|	Поразрядное ИЛИ	
11	&&	Логическое И	
12	||	Логическое ИЛИ	
13	? :	Условная	
14	= *= /= %= += -= &= |= >>= <<= ^=	Простое и составное присваивание	Справа налево
15	,	Последовательное вычисление	Слева направо

2.3.3. Преобразование типов

При выполнении операций происходят неявные преобразования типов в следующих случаях:

- при выполнении операций осуществляются обычные арифметические преобразования (которые были рассмотрены выше);
- при выполнении операций присваивания, если значение одного типа присваивается переменной другого типа;
- при передаче аргументов функции.

Кроме того, в Си есть возможность явного приведения значения одного типа к другому.

В операциях присваивания тип значения, которое присваивается, преобразуется к типу переменной, получающей это значение. Допускается преобразование целых и плавающих типов, даже если такое преобразование ведет к потере информации.

Преобразование целых типов со знаком. Целое со знаком преобразуется к более короткому целому со знаком посредством усечения старших битов. Целое со знаком преобразуется к более длинному целому со знаком путем размножения знака. При преобразовании целого со знаком к целому без знака целое со знаком преобразуется к размеру целого без знака и результат рассматривается как значение без знака.

Преобразование целого со знаком к плавающему типу происходит без потери информации, за исключением случая преобразования значения типа long int или unsigned long int к типу float, когда точность часто может быть потеряна.

Преобразование целых типов без знака. Целое без знака преобразуется к более короткому целому без знака или со знаком путем усечения старших битов. Целое без знака преобразуется к более длинному целому без знака или со знаком путем дополнения нулей слева.

Когда целое без знака преобразуется к целому со знаком того же размера, битовое представление не изменяется. Поэтому значение, которое оно представляет, изменяется, если знаковый бит установлен (равен 1), т.е. когда исходное целое без знака больше, чем максимальное положительное целое со знаком, такой же длины.

Целые значения без знака преобразуются к плавающему типу путем преобразования целого без знака к значению типа signed long, а затем значение

signed long преобразуется в плавающий тип. Преобразования из unsigned long к типу float, double или long double производятся с потерей информации, если преобразуемое значение больше, чем максимальное положительное значение, которое может быть представлено для типа long.

Преобразования плавающих типов. Величины типа float преобразуются к типу double без изменения значения. Величины double и long double преобразуются к float с некоторой потерей точности. Если значение слишком велико для float, то происходит потеря значимости, о чем сообщается во время выполнения.

При преобразовании величины с плавающей точкой к целым типам она сначала преобразуется к типу long (дробная часть плавающей величины при этом отбрасывается), а затем величина типа long преобразуется к требуемому целому типу. Если значение слишком велико для long, то результат преобразования не определен.

Преобразования из float, double или long double к типу unsigned long производятся с потерей точности, если преобразуемое значение больше, чем максимально возможное положительное значение, представленное типом long.

Преобразование типов указателя. Указатель на величину одного типа может быть преобразован к указателю на величину другого типа. Однако результат может быть не определен из-за отличий в требованиях к выравниванию и размерах для различных типов.

Указатель на тип void может быть преобразован к указателю на любой тип, и указатель на любой тип может быть преобразован к указателю на тип void без ограничений. Значение указателя может быть преобразовано к целой величине. Метод преобразования зависит от размера указателя и размера целого типа следующим образом:

- если размер указателя меньше размера целого типа или равен ему, то указатель преобразуется точно так же, как целое без знака;
- если указатель больше, чем размер целого типа, то указатель сначала преобразуется к указателю с тем же размером, что и целый тип, и затем преобразуется к целому типу.

Целый тип может быть преобразован к адресному типу по правилам:

- если целый тип того же размера, что и указатель, то целая величина просто рассматривается как указатель (целое без знака);

- если размер целого типа отличен от размера указателя, то целый тип сначала преобразуется к размеру указателя (используются способы преобразования, описанные выше), а затем полученное значение трактуется как указатель.

Преобразования при вызове функции. Преобразования, выполняемые над аргументами при вызове функции, зависят от того, был ли задан прототип функции (объявление "вперед") со списком объявлений типов аргументов.

Если задан прототип функции и он включает объявление типов аргументов, то над аргументами в вызове функции выполняются только обычные арифметические преобразования.

Эти преобразования выполняются независимо для каждого аргумента. Величины типа float преобразуются к double, величины типа char и short - к int, величины типов unsigned char и unsigned short - к unsigned int. Могут быть также выполнены неявные преобразования переменных типа указатель. Задавая прототипы функций, можно переопределить эти неявные преобразования и позволить компилятору выполнить контроль типов.

Преобразования при приведении типов. Явное преобразование типов может быть осуществлено посредством операции приведения типов, которая имеет формат:

(имя-типа) операнд;

В приведенной записи имя-типа задает тип, к которому должен быть преобразован операнд.

Пример:

```
int    i=2;  long    p=2;
double    d;  float     f;
d=(double)i * (double)p;
f=(float)d;
```

В данном примере величины i,p,d будут явно преобразовываться к указанным в круглых скобках типам.

ЛИТЕРАТУРА

1. Единая система программной документации.- М: издательство стандартов, 1998.-164 с.

2. Керниган Б., Ритчи Д. Язык программирования Си: Пер. с англ./ Под ред. и с предисл. Вс. С. Штаркмана. – 2-е изд., перераб. и доп. – М.: Финансы и статистика, 1992. – 272 с.

3. Кнут Д. Искусство программирования. Т.1-3.-М: Мир, 1976-1978.

4. Юлин В.А., Булатова И.Р. Приглашение к СИ. – Мн: Выш. шк., 1990.-224 с.