

**Владимирский государственный университет**

**М. В. ШИШКИНА**

**ОСНОВЫ АЛГОРИТМИЗАЦИИ  
И ПРОГРАММИРОВАНИЯ**

**Практикум**

**Владимир 2023**

Министерство науки и высшего образования Российской Федерации  
Федеральное государственное бюджетное образовательное учреждение  
высшего образования

«Владимирский государственный университет  
имени Александра Григорьевича и Николая Григорьевича Столетовых»

М. В. ШИШКИНА

# ОСНОВЫ АЛГОРИТМИЗАЦИИ И ПРОГРАММИРОВАНИЯ

Практикум

*Электронное издание*



Владимир 2023

ISBN 978-5-9984-1836-5  
© Шишкина М. В., 2023

УДК 004.42  
ББК 32.973-018

Рецензенты:

Кандидат технических наук  
генеральный директор ООО «ФС Сервис»  
*Д. С. Квасов*

Кандидат физико-математических наук  
зав. кафедрой функционального анализа и его приложений  
Владимирского государственного университета  
имени Александра Григорьевича и Николая Григорьевича Столетовых  
*В. Д. Бурков*

**Шишкина, М. В.**

Основы алгоритмизации и программирования [Электронный ресурс] : практикум / М. В. Шишкина ; Владим. гос. ун-т им. А. Г. и Н. Г. Столетовых. – Владимир : Изд-во ВлГУ, 2023. – 111 с. – ISBN 978-5-9984-1836-5. – Электрон. дан. (1,66 Мб). – 1 электрон. опт. диск (CD-ROM). – Систем. требования: Intel от 1,3 ГГц ; Windows XP/7/8/10 ; Adobe Reader ; дисковод CD-ROM. – Загл. с титул. экрана.

Приведены теоретические аспекты с разбором достаточного количества примеров, представлены задания к лабораторным работам, упражнения для самостоятельной работы и контрольные вопросы для самоподготовки и закрепления материала.

Предназначен для студентов бакалавриата направлений подготовки 12.03.05 «Лазерная техника и лазерные технологии» и 28.03.01 «Нанотехнологии и микросистемная техника».

Рекомендовано для формирования профессиональных компетенций в соответствии с ФГОС ВО.

Ил. 51. Табл. 2. Библиогр.: 5 назв.

ISBN 978-5-9984-1836-5

© Шишкина М. В., 2023

## ОГЛАВЛЕНИЕ

ПРЕДИСЛОВИЕ .....	4
ВВЕДЕНИЕ .....	5
1. ОСНОВНЫЕ АЛГОРИТМИЧЕСКИЕ СТРУКТУРЫ.....	6
Лабораторная работа № 1 .....	19
2. БАЗОВЫЕ ТИПЫ ЯЗЫКА C++ .....	23
Лабораторная работа № 2 .....	29
3. ОПЕРАТОРЫ ВЕТВЛЕНИЯ ЯЗЫКА C++ .....	36
Лабораторная работа № 3 .....	41
4. ОПЕРАТОРЫ ОРГАНИЗАЦИИ ЦИКЛОВ ЯЗЫКА C++ .....	44
Лабораторная работа № 4 .....	51
5. УКАЗАТЕЛИ И ССЫЛКИ. МАССИВЫ. ДИНАМИЧЕСКОЕ РАСПРЕДЕЛЕНИЕ ПАМЯТИ.....	54
5.1. УКАЗАТЕЛИ И ССЫЛКИ .....	54
5.2. СТАТИЧЕСКИЕ МАССИВЫ .....	60
5.3. МНОГОМЕРНЫЕ МАССИВЫ.....	67
5.4. ДИНАМИЧЕСКОЕ РАСПРЕДЕЛЕНИЕ ПАМЯТИ .....	69
Лабораторная работа № 5 .....	72
6. СПОСОБЫ СОРТИРОВКИ МАССИВОВ .....	75
6.1. СОРТИРОВКА ВЫБОРОМ .....	75
6.2. СОРТИРОВКА ПУЗЫРЬКОМ .....	77
6.3. СОРТИРОВКА ПРОСТОЙ ВСТАВКОЙ .....	79
Лабораторная работа № 6 .....	80
7. ПОЛЬЗОВАТЕЛЬСКИЕ ТИПЫ ДАННЫХ. СТРУКТУРЫ.....	82
Лабораторная работа № 7 .....	84
8. ФУНКЦИИ .....	87
Лабораторная работа № 8 .....	92
9. ЛИНЕЙНЫЕ ДИНАМИЧЕСКИЕ СТРУКТУРЫ ДАННЫХ .....	94
Лабораторная работа № 9 .....	105
ЗАКЛЮЧЕНИЕ.....	108
РЕКОМЕНДАТЕЛЬНЫЙ БИБЛИОГРАФИЧЕСКИЙ СПИСОК .....	109
ПРИЛОЖЕНИЕ .....	110

## ПРЕДИСЛОВИЕ

Современное образование в любой области немыслимо без знания основ информационных технологий и грамотного владения современными прикладными программами.

Полноценное овладение профессией невозможно без изучения основ алгоритмизации и программирования, давно и прочно вошедших во все сферы нашей жизни. Базовые знания алгоритмизации и программирования дают обучающимся возможности дальнейшего познания различных аспектов современных информационных технологий. Освоение основ программирования и алгоритмизации позволяет обучающимся получить навыки самостоятельной разработки алгоритмических решений, создания и отладки кода, открывает возможности самостоятельно углублять свои знания по изученному языку программирования, осваивать другие языки программирования при построении индивидуального образовательного трека. Студенты, освоившие курс, могут принимать самостоятельное обоснованное решение при выборе средств разработки программного продукта для решения профессиональных задач. Важным является и приобретение навыков понимания кода, созданного другими разработчиками, поиска и исправления ошибок в нем.

Практикум содержит задания для лабораторных работ, упражнения для самостоятельной работы, вопросы для самоконтроля и подготовки к выполнению и защите лабораторных работ, входящих в состав учебно-методического комплекса дисциплины, разработанного на основе современных федеральных государственных образовательных стандартов, и разработан в соответствии с рабочими программами дисциплины «Основы алгоритмизации и программирования», может быть рекомендован в качестве основного учебного пособия для освоения дисциплины «Основы алгоритмизации и программирования».

## **ВВЕДЕНИЕ**

Практикум позволяет восстановить и расширить знания, усвоенные в ходе изучения школьного курса «Информатика и ИКТ», базируясь на полученных навыках, плавно перейти к углублённому изучению алгоритмизации и основ программирования на примере языка C++.

Практикум содержит девять разделов, дающих возможность закрепить знания, полученные на лекциях и практических занятиях, подготовиться и выполнению лабораторных работ.

Каждый раздел включает: информативную теоретическую часть с необходимым количеством примеров по рассматриваемой теме, цель и постановку задачи лабораторной работы, контрольные вопросы и упражнения для самостоятельной работы.

Контрольные задания подобраны таким образом, что позволяют обучающемуся повторить и сохранить в памяти материал рассматриваемой темы, без посторонней помощи объективно проанализировать свой уровень владения знаниями и сформированности навыков, а также подготовиться к защите лабораторной работы.

Упражнения для самостоятельной работы нацелены на закрепление полученных в процессе изучения курса знаний и навыков.

Практикум построен таким образом, что каждое следующее задание основано на навыках и знаниях, усвоенных при работе над предыдущим заданием.

В приложении к практикуму представлен образец титульного листа отчёта по лабораторной работе.

## 1. ОСНОВНЫЕ АЛГОРИТМИЧЕСКИЕ СТРУКТУРЫ

В IX веке жил и работал арабский учёный Аль-Хорезми. Среди широкого круга его интересов особое место занимали исследования в области математики. Аль-Хорезми впервые выделил алгебру как самостоятельную науку о методах решения линейных и квадратных уравнений, произвёл их классификацию уравнений, разработал подробные тригонометрические таблицы, осуществил подробные расчёты позиций Солнца, Луны и планет, солнечных затмений. Астрономические таблицы Аль-Хорезми были переведены на европейские, а затем и, китайский, языки. Аль-Хорезми проводил большую работу по популяризации науки. К сожалению, не дошёл до наших дней его трактат в котором, была впервые описана позиционная десятичная система счисления и сформулированы правила вычисления в ней. Имя Аль-Хорезми стало нарицательным, так стали называть вычисление по строго определённым правилам, а затем и любые строго предписанные действия, направленные на достижение определённой цели.

**Алгоритм** – четкое предписание исполнителю выполнить точно определенную последовательность действий, направленных на достижение заданной цели.

Исполнителем алгоритма может быть человек или автоматическое устройство, способное воспринимать запись алгоритма.

**Свойства алгоритма**, отличающие его от обычной последовательности действий:

1. **Дискретность** - разбиение алгоритма на последовательность шагов, отдельных законченных действий.

Каждый такой шаг должен быть завершён исполнителем до того, как он приступит к выполнению следующего шага.

2. **Точность** - однозначность указаний.

На каждом шаге однозначно определено состояние объектов среды исполнителя, полученной от предыдущих действий алгоритма. На каждом шаге алгоритма однозначно определено, какую команду надо выполнять следующей. Таким образом, при многократном применении алгоритма к одному и тому же набору входных данных, на выходе каждый раз должен быть получен один и тот же результат.

3. *Понятность* - однозначная трактовка исполнителем каждого шага алгоритма.

Алгоритм должен быть изложен на языке понятном для исполнителя, т.е., состоять из команд, входящих в систему команд исполнителя (СКИ).

4. *Конечность* (результативность) - обязательное получение результата за конечное число шагов.

Каждый шаг (и алгоритм в целом) после своего завершения дает среду, в которой все объекты однозначно определены. Если получение результата невозможно, пользователь должен получить сообщение, о том, что решение задачи не существует.

Работа алгоритма должна быть завершена за конечное число шагов, в любом случае, даже если решение не найдено. Теоретические аспекты бесконечных алгоритмов в этом курсе не изучаются.

5. *Массовость* - применение алгоритма к решению всего класса однотипных задач.

### ***Способы представления алгоритма***

Выделяют следующие формы записи алгоритмов:

- 1) графическая запись (блок-схемы);
- 2) на естественном языке (словесная запись, псевдокод);
- 3) код на языке программирования;
- 4) в виде математической формулы.

Запись алгоритма на естественном языке или словесная форма представления алгоритма - это описание на естественном языке последовательности шагов решения задачи. Такой подход широко применяется в повседневной жизни и крайне редко используется в профессиональной деятельности, отсутствие строгой формализованности описаний может привести к расхождениям в понимании инструкций.

Псевдокодом называют полужформализованное описание алгоритма, состоящее из элементов языка программирования, фраз, естественного языка, общепринятых математические обозначения и т.п.

Блок-схема или графическая форма записи, представляет собой отображение алгоритма в виде последовательности связанных между собой функциональных блоков, каждый такой блок соответствует выполнению одного или нескольких действий – шагов алгоритма. Блоки соединены между собой линиями, указывающими, какое действие бу-



дет выполнено следующим. Такая форма записи алгоритма более компактная и наглядная по сравнению со словесной.

Представление алгоритма в виде-блок схемы строго формализовано. Инструкции к представлению алгоритма таким образом содержатся в ГОСТ 19.701-90. ЕСПД. Схемы алгоритмов, программ данных и систем. Обозначения условные и правила выполнения.

Требования к форме и размерам блоков, изложены в ГОСТ 19.003-80. Схемы алгоритмов и программ. Обозначения условные и графические.

При изображении блоков, размер стороны  $a$  выбирается из ряда 15, 10, 20 мм, допускается увеличение значения параметра  $a$  на число кратное 5. Сторона  $b$  соотноситься со стороной  $a$  таким образом, что  $b=1.5a$ .

На рис. 1. Представлен блок обозначающий начало и конец алгоритма, вход во внешнюю среду и выход из среды. Обычно этот блок имеет следующее содержание «начало»/«конец», «запуск»/«останов», «перезапуск». Соответственно такой блок будет использован в блок-схеме два раза первым и завершающим. В рамках курса «Основы алгоритмизации и программирования» предполагается, что все блок-схемы учебных задач возможно, необходимо, оформить таким образом, что первый блок схемы содержит слово «Начало», последний «Конец». Использование блока, представленного на рис.1 в блок-схемах всех учебных задач, обязательно.



*Рис. 1. Блок, обозначающий начало и конец алгоритма*

Между блоками, о которых речь шла выше располагается последовательность блоков, соединенных линиями, такие линии могут содержать стрелки, указывающие направление, т.е. по стрелке можно определить какое действие будет выполнено следующим. Общепринятым считается направление сверху вниз и слева направо.

В таблице 1 кратко представлено описание основных блоков, используемых при оформлении алгоритмического решения в виде блок-схемы. Таблица содержит две колонки. В левой изображены

формы блоков с указанием сторон  $a$  и  $b$ , о соотношении сторон было упомянуто выше. Правая колонка таблицы содержит названия блоков и описание их назначения.

Таблица 1

### Основные блоки

Форма блока	Назначение блока
	<p><b>Блок – терминатор</b> Этот блок означает вход и выход во внешнюю среду. «Начало», «конец», или прерывание обработки данных.</p>
	<p><b>Блок – процесс</b> В этом блоке указывают одно или несколько действий, выполнение которых ведет к изменению данных, их значения или формы хранения.</p>
	<p><b>Блок - предопределённый процесс</b> Используют при необходимости использования ранее созданных и отдельно описанных алгоритмов, например, при вызове описанных ранее функций.</p>
	<p><b>Блок – данные (ввод-вывод)</b> Используют при необходимости ввода или вывода данных. Преобразование данных в форму, подходящую для обработки (ввод) или отображения результатов обработки (вывод).</p>
	<p><b>Блок – решение (условие) или переключатель.</b> Позволяет выбрать одну из двух возможных ветвей алгоритма. Блок имеет один вход и два альтернативных выхода. В блоке содержится некоторое условие, в зависимости от его значения будет определён выход из блока и дальнейшее направление алгоритма. Значения условия могут быть подписаны рядом с соответствующими выходами.</p>

Форма блока	Назначение блока
	<p>Блок – <b>подготовка</b> (цикл с параметром).          Может содержать установку переключателя, модификацию индекса.</p>
<p>Начало цикла</p>  <p>Конец цикла</p> 	<p>Блок – <b>граница цикла</b>          Этот блок используют для обозначения границ цикла. Блок состоит из двух частей, отображающих начало и конец цикла соответственно.          Условия инициализации, приращения, завершения цикла помещают в начале или конце цикла в зависимости от того это цикл с предусловием или постусловием.</p>
	<p>Блок – <b>соединитель</b>.          Используют при необходимости разрыва линии и продолжении её в другом месте. Например, если блок-схема не помещается на страницу. Блок с помещённым в него уникальным символом размещают в моменте обрыва и возобновления линии.</p>
	<p><b>Комментарий</b>          Используют для описания и/или разъяснения каких-то действий.</p>
	<p><b>Пропуск</b>          Три точки используют для отображения пропущенного символа или группы символов</p>

Часто возникает необходимость пояснить назначение какого-то фрагмента схемы.

Пример такого пояснения представлен на рис. 2. Комментируемую группу блоков заключают в пунктирную рамку. Сам комментарий размещают в прямоугольной скобке, которую соединяют с рамкой пунктиром.

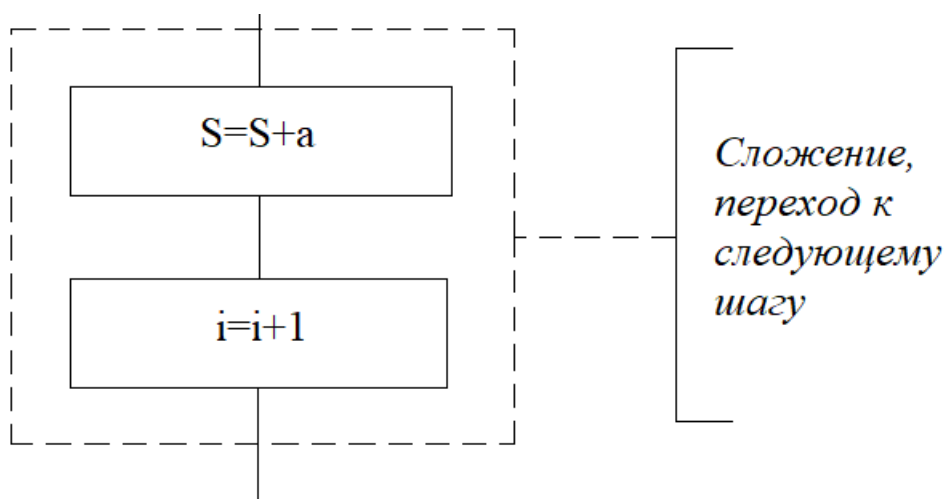


Рис. 2. Использование комментариев в блок-схеме

Если решение задачи довольно сложное и объёмное, то представление его в виде блок-схемы может получиться слишком громоздким. Для избежания такой ситуации определённую последовательность блоков, решающих отдельную подзадачу, заменяют блоком: «предопределённый процесс», содержащим обращение к ранее разработанному и описанному алгоритму, например. Заменяемый блок не должен быть элементарен (т.е. должен содержать совокупность блоков). Других ограничений на количество заменяемых блоков нет. Существует только одно требование – больше одного блока, полное решение конкретной подзадачи. Если такая задача решена правильно, то заменяемая совокупность блоков будет реализовывать определённый алгоритм, соответственно иметь все признаки алгоритма.

В графическом представлении алгоритма блоки соединены друг с другом линиями только через точки входа и выхода, таким образом, если задача внутри блока решена верно, то его внутренняя структура не влияет на остальную часть алгоритма.

Таким образом блок «предопределённый процесс» может быть включён в блок-схему до того, как будет разработан сам алгоритм решения подзадачи.

Линии, соединяющие блоки могут быть сопровождаемы стрелками, указывающими направление, если оно отлично от принятого направления: сверху вниз и слева направо. В любом случае линии должны быть направлены к центру символа и образовывать прямой угол со стороной прямоугольника, если блок не прямоугольник, то он может быть вписан в прямоугольник.

При работе над блок-схемой нужно стремиться избегать большого числа длинных линий, стараться минимизировать количество пересечений. Если пересечения избежать невозможно, они должны быть строго под прямым углом, в точке пересечения линия не может менять направление. В случае большого количества пересечений и длинных линий

рекомендуется линии разрывать и использовать соединители.

При объединении линий, место слияния обозначают точкой. Примеры слияния линий представлены на рис. 3.

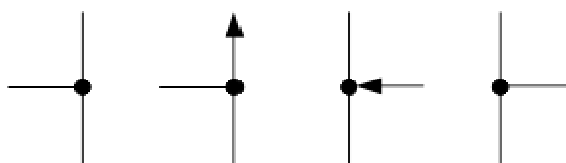


Рис. 3. Слияние линий потока

Внутри блока размещают текст минимально необходимый для понимания содержания блока.

Текст записывается слева направо и сверху вниз. Если весь нужный текст не представляется возможным поместить в блок, текст выносят в комментарии.

Рассмотрим три основные алгоритмические структуры (виды алгоритмов): следование (линейный), ветвление (разветвляющийся) и цикл (циклический алгоритм). Алгоритмы, содержащие несколько алгоритмических структур, называют комбинированными.

**Следование** – это алгоритмическая структура, в которой все команды выполняются последовательно, одна за другой. Блок-схема такой конструкции, представляет собой последовательность блоков: «Процесс». Структурная схема такой конструкции приведена на рис.4. Алгоритмы, в которых используется только алгоритмическая конструкция «следование», называются *линейными алгоритмами*.

Используя только алгоритмическую структуру *следование* можно решить ряд простейших задач, например, ввести с клавиатуры радиус круга и вычислить длину дуги окружности. Примеры блок-схем, реализованных только с использованием алгоритмической структуры *следование* приведены на рис.5.

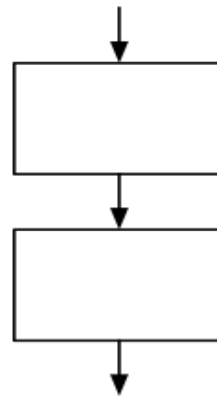
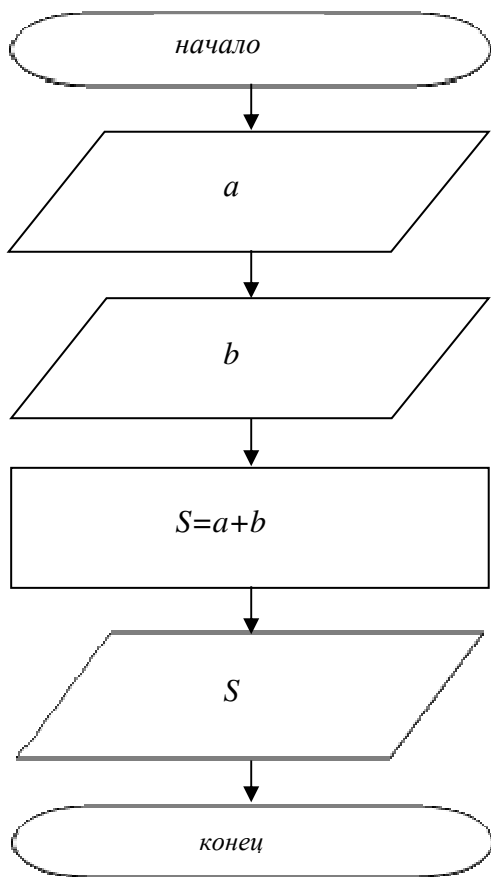
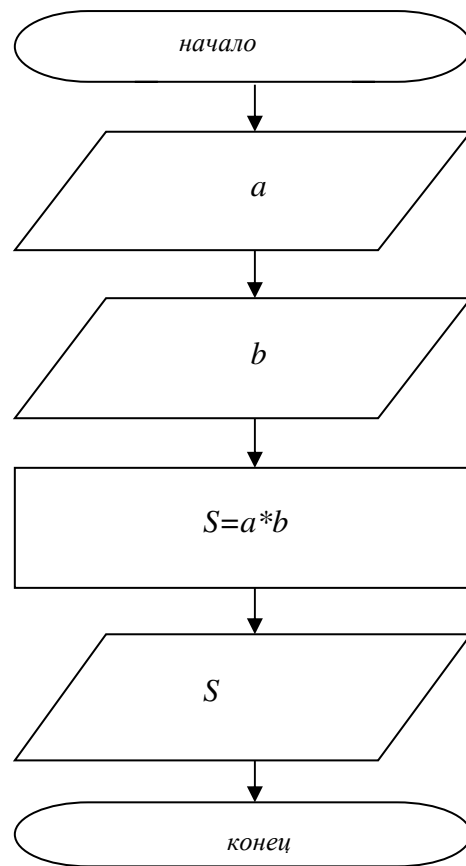


Рис. 4. Схематичное представление алгоритмической структуры *следование*



а)



б)

Рис. 5. Блок схемы линейных алгоритмов.  
*а* – вычисление суммы двух чисел., *б* – вычисление площади прямоугольника

**Ветвление** – алгоритмическая структура, в которой выполнение определённого действия зависит от значения условия, будет выполняться либо одна, либо другая последовательность действий.

Таким образом, структура ветвление включает в себя условие и одну или две последовательности команд.

Существует две формы «ветвления»: полная и неполная. В полной форме ветвление, содержит два действия (последовательности команд), одно из этих действий будет выполнено при значении условия – «истина», а второе при значении условия – «ложь». В неполной форме ветвление содержит только одно действие или последовательность команд, которые будут выполнены при значении условия – «истина». На рис. 6 приведены структурные схемы полной и неполной форм ветвления.

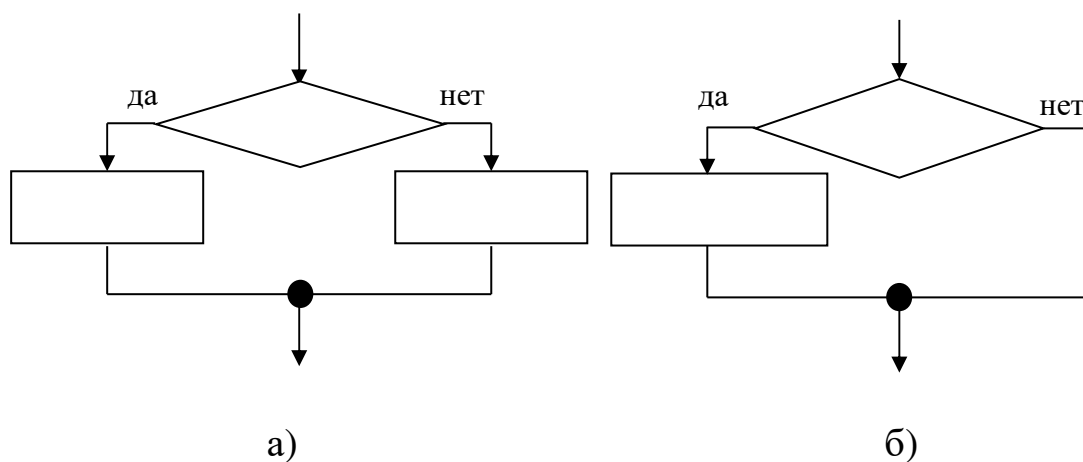


Рис. 6. Алгоритмическая структура «ветвление»:  
а - полное ветвление б - неполное ветвление

Алгоритмы, в основе которых лежит алгоритмическая конструкция: «ветвление», называют *разветвляющимися*.

На рис. 7 приведены примеры блок-схем таких алгоритмов, определение наибольшего из двух чисел, и получение модуля числа.

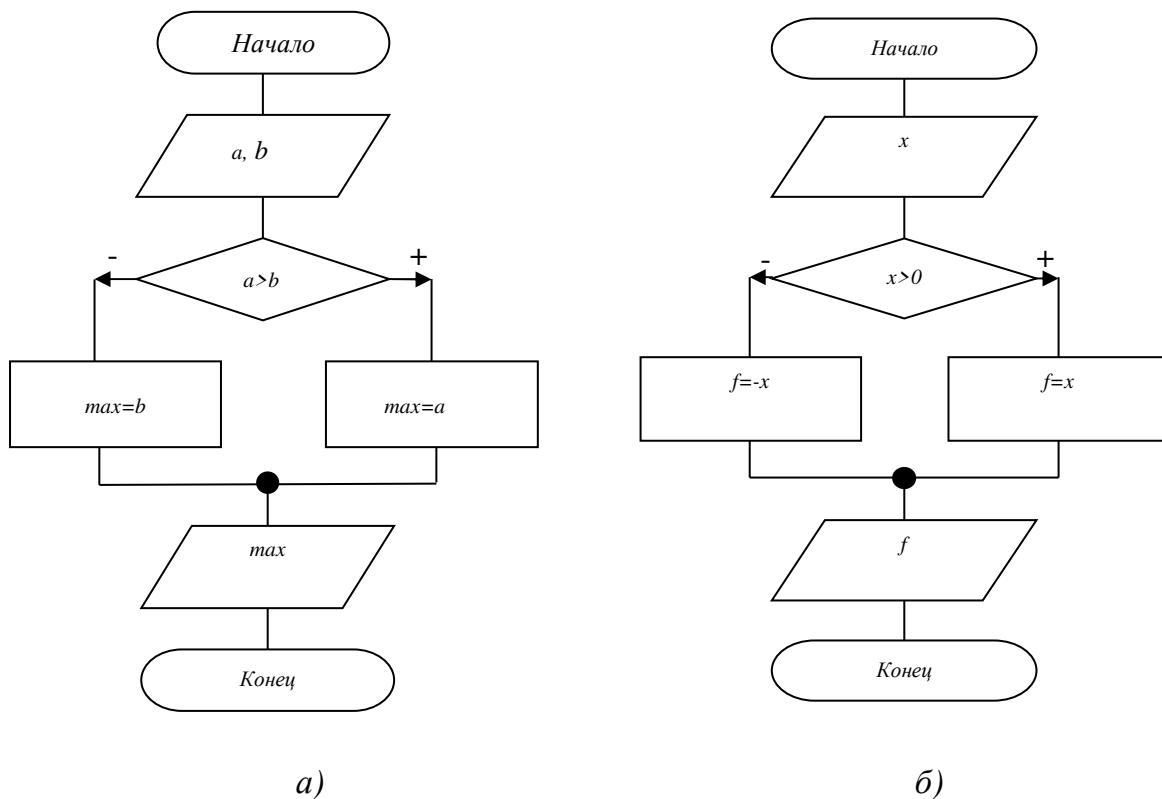


Рис. 7. Разветвляющие алгоритмы: а - поиск максимального значения, б - модуль числа

Определим принадлежит ли точка  $x$  отрезку  $[a, b]$ . Для этого необходимо проверить два условия:  $x \geq a$  и  $x \leq b$ . Для проверки этих условий можно использовать две условные конструкции. А можно одну объединив условия в одно с помощью логической операции «и».

Блок-схема этой части алгоритма представлена на рис. 7.

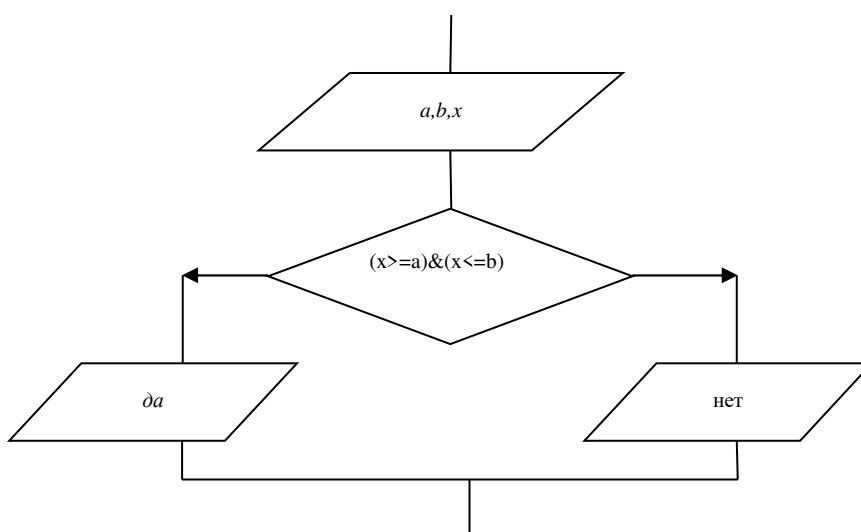


Рис. 8. Алгоритм определения принадлежности точки отрезку



На рис. 9 показана структурная схема вложенного ветвления.

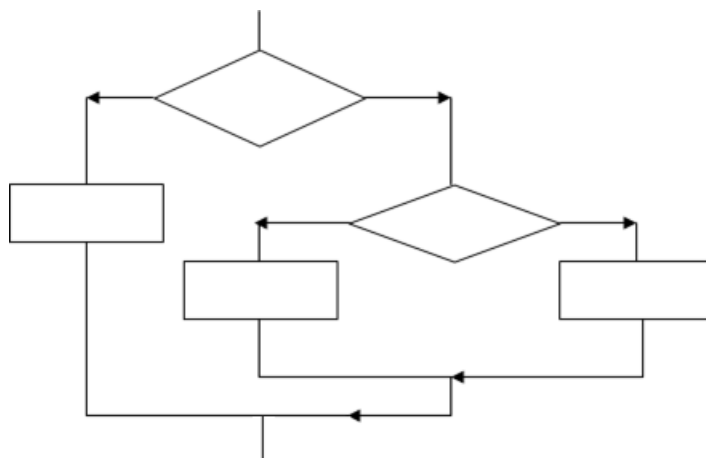


Рис. 9. Структурная схема вложенного ветвления

**Повторение** (цикл) – алгоритмическая конструкция, с помощью которой определенная последовательность действий выполнится необходимое число раз. Алгоритмы, основой, которых служит конструкция повторение называют циклическими или *циклом*.

*Телом цикла* называют действия многократно повторяющиеся в процессе выполнения цикла.

Выделяют два типа циклов (по взаимному расположению тела цикла и условия продолжения):

- 1) цикл с *постусловием*;
- 2) цикл с *предусловием*.

При использовании конструкции «цикл с предусловием» проверка условия происходит до выполнения действий тела цикла, таким образом возможна ситуация, в которой тело цикла не выполнится ни одного раза.

При использовании конструкции: «цикл с постусловием», тело цикла будет выполнено как минимум один раз, так как проверка условия осуществляется после выполнения тела цикла и в зависимости от результата этой проверки будет осуществлён выход из цикла или переход на следующую итерацию.

Структурная схема цикла с предусловием и цикла с пост условием представлена на рис. 10.

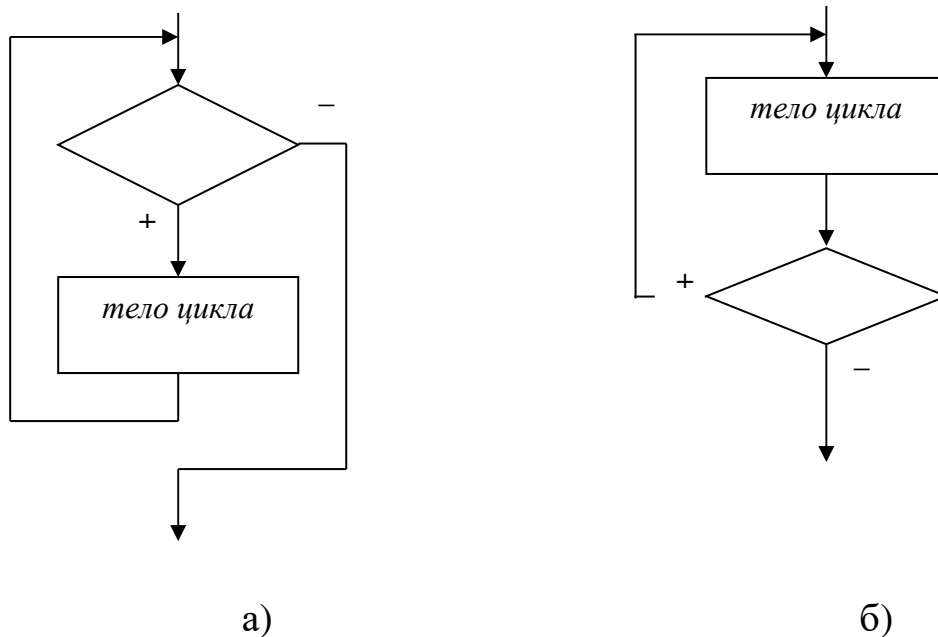


Рис. 10 Алгоритмическая конструкция «повторение»: а – цикл с предусловием, б – цикл с постусловием

Для успешной организации алгоритмической конструкции «повторение», следует до входа в цикл задать начальные значения переменных, используемых в цикле. В теле цикла необходимо предусмотреть изменение переменных, анализируемых в условии продолжения цикла.

При решении конкретных задач алгоритм может содержать более одной конструкции «повторение». В зависимости от их взаимного расположения говорят о вложенных или последовательных циклах.

Если один цикл является частью тела другого цикла, то первый цикл называют вложенным, второй – внешним.

На рис. 11 схематично показаны способы группировки конструкций повторение.

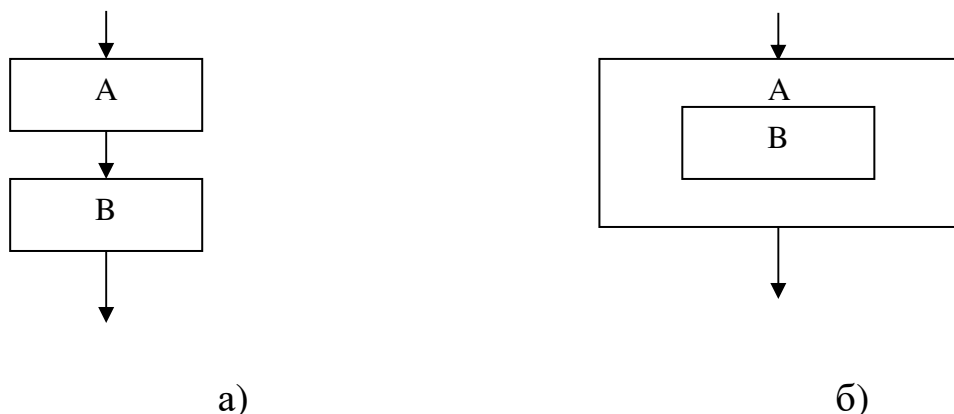


Рис. 11. Алгоритмическая конструкция «повторение»: а – последовательные циклы, б – вложенный цикл

Перед решением задачи и написанием кода на языке программирования необходимо представить графическое решение в виде блок-схемы.

Алгоритмы составленные на основе более чем одной алгоритмической конструкции называют комбинированными. Разберём пример такой задачи.

Вычислим произведение некоторого количества вводимых с клавиатуры целых чисел. Количество чисел зададим параметром  $n$ . В сумме будем учитывать только числа большие нуля.

На рис. 12 представлена блок-схема решения этой задачи.

До входа в цикл в блоке «действие» значение следующих переменных  $n$  - количество слагаемых,  $i$  – счётчик,  $P$  – произведение. Для получения корректного результата важно начальное значение произведения равным единицы.

Рассмотрим подробнее организацию цикла. Для решения задачи, использован цикл с постусловием. Т.е. тело цикла будет выполнено как минимум один раз. Ввод значения переменной  $x$  – первое действие тела цикла. Далее осуществляется проверка этого значения. Если значение переменной  $x$  больше нуля, производим вычисление произведения и увеличение значения счётчика. В противном случае не производим никаких действий. Тело цикла на этом завершено. Переходим к проверке условия. Сравниваем значение счётчика с необходимым числом множителей. Т.е. сравниваем значение переменной  $i$  и  $n$ . Если счётчик меньше или равен необходимому количеству множителей, переходим на следующую итерацию цикла. Иначе – выходим из цикла.

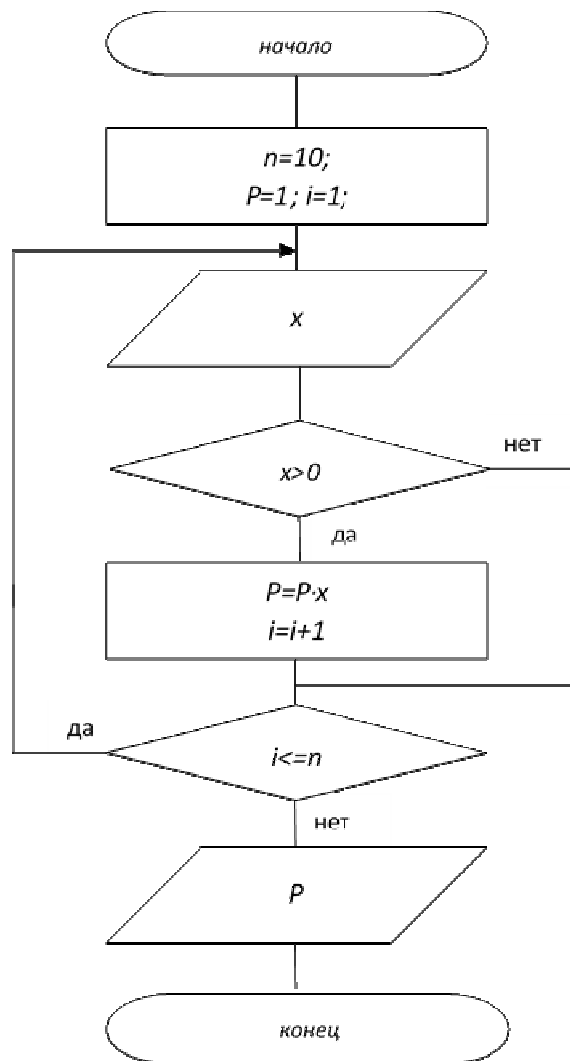


Рис. 12. Блок-схема алгоритма нахождения произведения  $n$ , введённых с клавиатуры чисел

## Лабораторная работа № 1 ЛИНЕЙНЫЕ АЛГОРИТМЫ

### Цель работы

Повторить, изученные в школьном курсе предмета «информатика и ИКТ», понятие алгоритма, свойства алгоритмов, способы представления алгоритмов. Получить и закрепить на практике навыки разработки линейных алгоритмов. Представления разработанных алгоритмов графически и на языке программирования C++.

### ***Постановка задачи***

1. Вычислить сумму двух введенных с клавиатуры вещественных чисел.  $S=a+b$ , где  $a$  и  $b$  вещественные числа.

2. Даны длины сторон треугольника  $A$ ,  $B$ ,  $C$ . Найти площадь треугольника  $S$ .

3. Даны координаты вершин треугольника  $ABC$ . Найти его площадь. Составьте блок-схему алгоритма решения поставленной задачи.

4. В квадратной комнате шириной  $A$  и высотой  $B$  есть окно и дверь с размерами  $C$  на  $D$  и  $M$  на  $N$  соответственно. Вычислите площадь стен для оклеивания их обоями. Составьте блок-схему алгоритма решения поставленной задачи.

5. Дана величина  $A$ , выражающая объем информации в байтах. Перевести  $A$  в более крупные единицы измерения информации. Составьте блок-схему алгоритма решения поставленной задачи.

### ***Вопросы для самоконтроля, подготовки к выполнению и защите лабораторной работы***

1. Что такое алгоритм?
2. Какие существуют способы представления алгоритмических решений?
3. Перечислите достоинства и недостатки описания алгоритма на естественном языке.
4. Перечислите свойства алгоритма.
5. Какие алгоритмы называют линейными?
6. Для чего в блок-схемах используют комментарии?
7. Для чего при графическом представлении алгоритма используют блок «терминатор»?
8. Для чего в блок-схемах используют блок соединитель?
9. Что в блок-схеме означают стрелки?
10. Назовите соотношение сторон  $a$  и  $b$  основных блоков блок-схемы, оформленной в соответствии с ГОСТ 19.003-80.
11. Каково назначение блока «процесс»?
12. Для каких целей используют блок «Предопределённый процесс»?
13. Что нужно сделать при необходимости перенести оформление блок-схемы на другую страницу?

## Задания для самостоятельной работы

1. Вычислить путь, пройденный лодкой, если ее скорость в стоячей воде  $v$  км/ч, скорость течения реки  $v1$  км/ч, время движения по озеру  $t1$  ч, а против течения реки –  $t2$  ч. Составьте блок-схему алгоритма решения поставленной задачи.

2. Вычислите значение функции  $Y$  при  $X=2$ , используя, представленную на рис. 13, а) блок-схему алгоритма

3. По блок-схеме вычисления значения некоторой функции, представленной на рис. 13, б) восстановите условие задачи; напишите формулу вычисления значения функции.

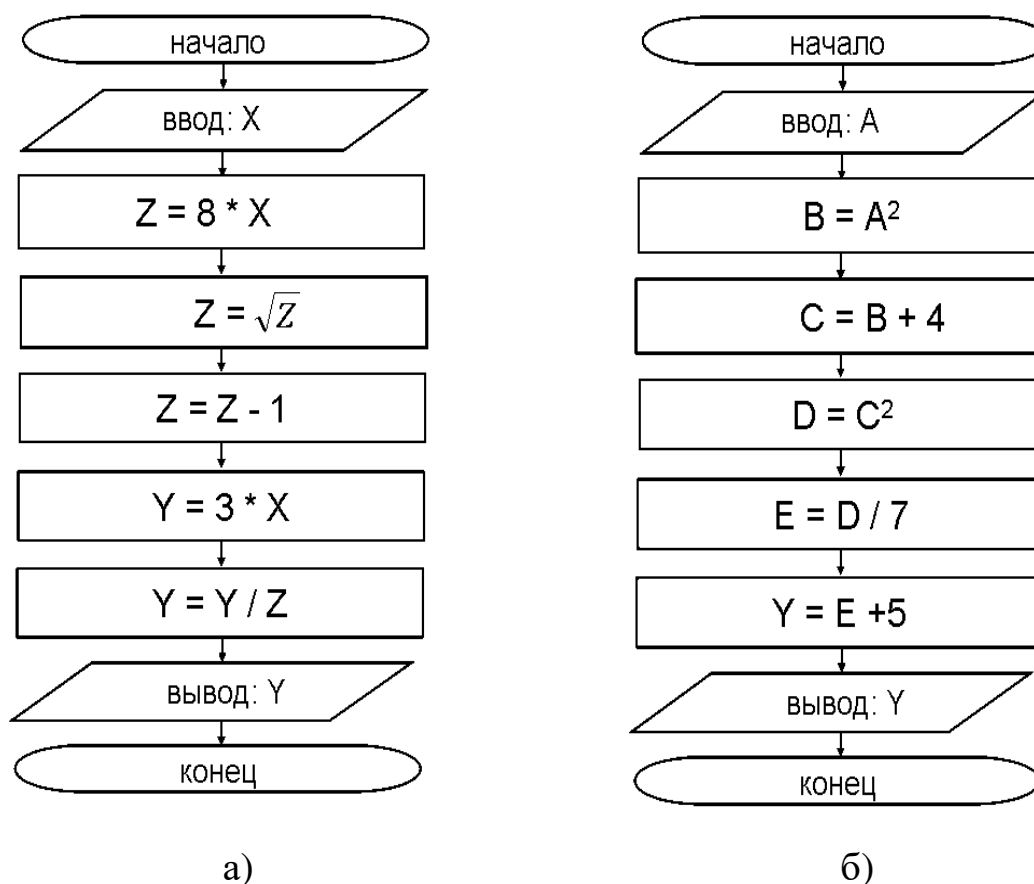


Рис. 13. Блок-схемы к заданиям для самостоятельной работы: а – для задачи 2, для задачи - 3

4. По блок-схеме вычисления значения некоторой функции, представленной на рис. 14, а) восстановите условие задачи; напишите формулу вычисления значения функции.

5. По блок-схеме вычисления значения некоторой функции, представленной на рис. 14, б) восстановите условие задачи; напишите формулу вычисления значения функции.

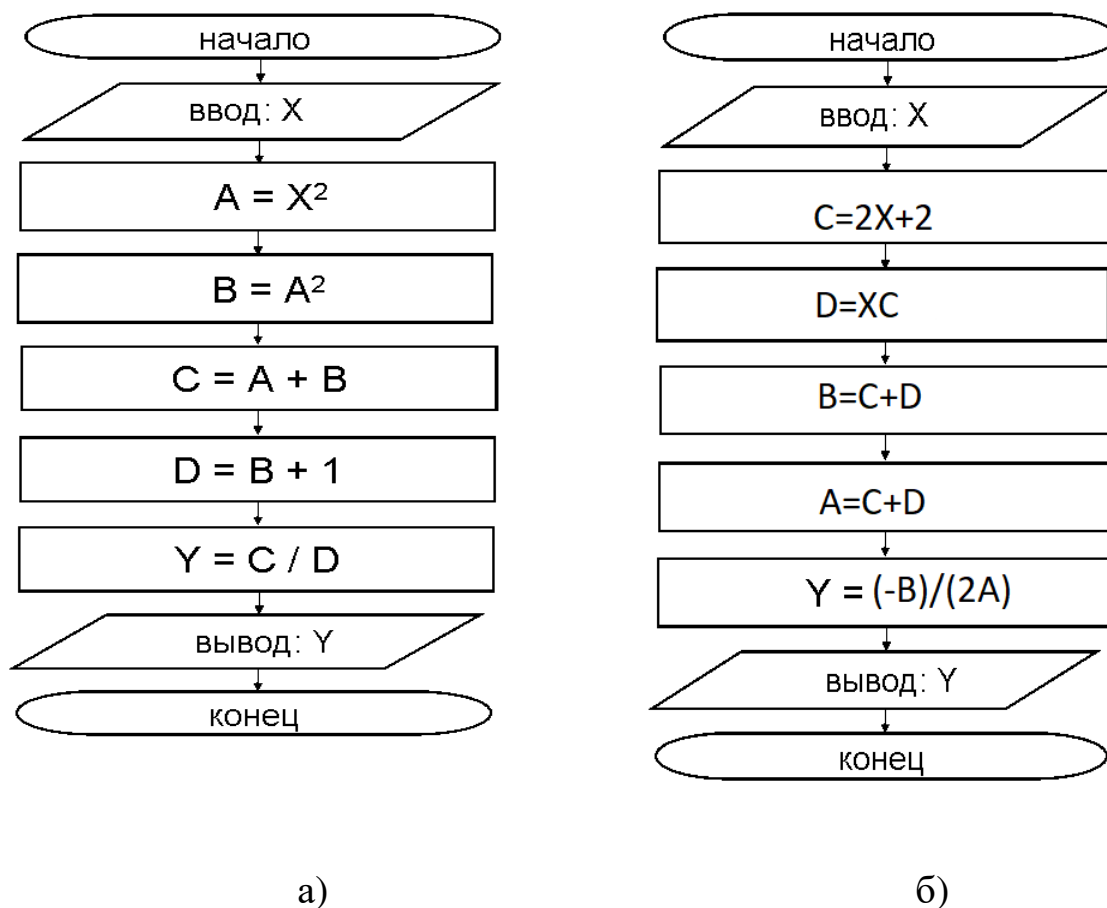


Рис. 14. Блок-схемы к заданиям для самостоятельной работы: а – для задачи 4, для задачи – 5

6. По двум введенным пользователем катетам вычислить длину гипотенузы.

7. Пользователь вводит две буквы. Определить, на каких местах алфавита они стоят, и сколько между ними находится букв.

8. Пользователь вводит номер буквы в алфавите. Определить, какая это буква.

9. Пользователь вводит значения двух целочисленных переменных. Необходимо поменять значения переменных так, чтобы значение первой оказалось во второй, а второй — в первой.

10. Вывести на экран вещественные числа с заданным количеством знаков после запятой.
11. Пользователь вводит длину и ширину прямоугольника, вычислить и вывести на экран его площадь и периметр.
12. Пользователь вводит радиус круга, вычислить и вывести на экран его площадь и длину дуги окружности.
13. Рассчитать ежемесячные выплаты по кредиту и суммарную выплату. Данные о кредите вводит пользователь. Кредит составляет  $n$  рублей, берется на  $u$  лет, под  $p$  процентов.
14. Пользователь вводит трёхзначное число. Найти сумму и произведение цифр этого числа.
15. Пользователь вводит координаты двух точек. Вывести уравнение прямой вида:  $y = kx + b$ .
16. Пользователь вводит два числа, выполнить над ними логические побитовые операции.
17. Даны длины ребер  $a$ ,  $b$ ,  $c$  прямоугольного параллелепипеда. Найти его объем и площадь поверхности.
18. Дана длина ребра куба  $a$ . Найти его объем и площадь поверхности.
19. Найти значение функции  $y = 3x^6 - 6x^2 - 7$  при данном значении  $x$ .
20. Найти значение функции  $y = 4(x-3)^6 - 7(x-3)^3 + 2$  при данном значении  $x$ .
21. Дано значение температуры  $T$  в градусах Цельсия. Определить значение этой же температуры в градусах Фаренгейта.

## 2. БАЗОВЫЕ ТИПЫ ЯЗЫКА C++

Перед объявлением переменной или константы необходимо определить какого она будет типа. Для этого нужно понимать цели для которых она будет использована, какие значения она может принимать.

*Тип данных* определяет количество выделяемой памяти под переменную или константу, правила хранения данных этого типа, допустимые операции для данных этого типа.



Диапазон значений, данных конкретного типа данных определяется количеством памяти, выделяемым под переменную и правилами хранения данных.

Типы данных языка C++ можно разделить на две группы: базовые и пользовательские.

К базовым типам языка относят: `void`, `int`, `char`, `float`, `double`, `bool`.

Кроме того, перед некоторыми типами могут быть указаны спецификаторы, влияющие на количество памяти, выделяемой под переменную и способ хранения данных.

Так, например, к типам `int`, `short`, `long`, `char`, применимы спецификаторы `signed` (знаковый), `unsigned` (беззнаковый).

Использование спецификаторов `short` и `long` перед типом данных `int` ведёт к уменьшению или увеличению количества выделяемой памяти.

Целые беззнаковые числа хранятся в прямом двоичном коде. Диапазон таких данных от нуля, до числа двоичное представление, которого состоит из ряда единиц, длина ряда – количество бит отводимых под число.

Таким образом диапазон беззнаковых целых типов данных будет определён следующим образом:  $0 \text{ — } 2^n - 1$ , где  $n$  – число разрядов отводимых под число.

0	0	0	0	0	0	0	0	0	0 минимальное значение
1	1	1	1	1	1	1	1	1	255 максимальное значение

Знаковые целые числа хранятся в дополнительном коде. Старший бит хранит знак. Единица – минус, ноль - плюс.

Незначащие разряды, следующие за знаковым, заполняются нулями. Например, если под переменную отведено 8 разрядов, то двоичное число 1001 в памяти будет представлено так:

0	0	0	0	1	0	0	1
---	---	---	---	---	---	---	---

Дополнительный код отрицательного числа  $m$  равен  $2^k - |m|$ , где  $k$  - количество разрядов в ячейке.



Для хранения символов набора из 256 символов ASCII применяют тип `char`. Для работы с символами, код, которых занимает более одного байта используют тип `wchar_t`.

Оператор `sizeof()`, возвращает количество памяти в байтах, занимаемое параметром. Например:

```
printf("int =%u\n", sizeof(int));
```

На экран будет выведено количество байт, выделяемое под данные типа `int`.

Поименованную область памяти, хранящую значение определённого типа называют *переменной*.

Для объявления переменной необходимо указать её тип и имя, завершив объявление точкой с запятой.

```
<имя типа> <имя переменной>;  
int i;
```

Можно совместить объявление переменной с инициализацией.

```
<имя типа имя переменной> = <значение>;  
int i = 25;
```

При необходимости объявить несколько одноименных переменных их разделяют запятой.

```
int i, j;
```

При объявлении константы перед её именем необходимо указать ключевое слово `const`. Значение константы нужно задать при объявлении.

```
<имя типа> const <имя константы> = <значение>;  
int const N=7;
```

### ***Правила именования в языке C++***

Любой идентификатор (имя) должен начинаться с латинской буквы. Кроме того, имя может содержать цифры и символ подчёркивания. Язык программирования C++ регистрочувствительный, это означает, что `int z;` и `int Z;` - объявление двух разных переменных. Имена переменных должны нести смысловую нагрузку т.е. из

названия переменной должно следовать. для каких целей она используется.

Место описания идентификатора задаёт его область действия.

Переменная описанная внутри блока будет локальной, т.е. видна только в этом блоке от точки объявления и ниже. Блоком называют часть кода, ограниченную фигурными скобками, например, тело цикла. Переменную, объявленную вне блока, называют глобальной. Обращение к такой переменной возможно в модуле, в котором она объявлена от точки объявления и ниже. Если имя локальной переменной и глобальной совпадают, то локальная перекрывает глобальную, т.е. обращение внутри блока будет вестись к локальной переменной.

Рассмотрим пример, работы с глобальными и локальными переменными.

```
int f()// описание функции
{int a = 5;//объявление локальной переменной
  printf("a=%i\n",a); return ++a;
}// завершение описания функции
int a = 17;//объявление глобальной переменной

int main()
{printf("a=%i\n", a);
int a = 1;// //объявление локальной переменной
printf("a=%i\n", f());
printf("a=%i\n", a);
for (int a=0;a<3;a++) {printf("a=%i\n",a ); }}
```

Результат работы программы будет таким:

```
a=17
a=5
a=6
a=1
a=0
a=1
a=2
```

Часто при вычислении выражений в качестве операндов используют разнотипные данные. В таких случаях используют приведение типов.

При выполнении операции присваивания тип правого операнда приводится к типу левого неявно, если это возможно. При приведении вещественного типа к целочисленному, дробная часть отбрасывается. При приведении целого типа к вещественному, добавляется нулевая дробная часть. Например:

```
float q = 7.34;
int w = q; //w=7
q=w; //q=7.00
```

Для явного приведения типа необходимо указать тип к значению которого приводится значение исходного типа данных. При этом важно понимать, что приведения типа осуществляется только в указанной точке, сама переменная: её значение и способ хранения данных при этом мне изменяется.

Явное приведение можно осуществить двумя способами. Если есть уверенность, что приведение пройдет без потери данных, используют подход языка C. Перед идентификатором тип которого приводится, указывают нужный тип в круглых скобках. Например:

```
float w = 4.35;
int r = (int)w;
```

В случае, когда есть уверенность, что переполнение не произойдет используют операцию оператор `static_cast`.

```
static_cast <тип> (<выражение>)
```

```
int a = 54;
```

```
char ch = i; // неявное преобразование
```

Такой подход приведет к предупреждающему сообщению во время компиляции, чтобы избежать этого лучше сделать так:

```
int a = 54;
```

```
char ch = static_cast<char>(a);
```

Рассмотрим несколько примеров решения задач.

Дано  $n$  клеток и  $m$  зайцев, посаженных по этим клеткам. Рассчитать максимальное количество зайцев, которое гарантированно окажется в одной клетке (по принципу Дирихле). Натуральные числа  $n$  и  $m$  вводит пользователь.

Решение будет выглядеть так, как представлено ниже.

```
int n, m, count = 0;
cin >> n >> m;
count = m / n;
    m %= n;
if (m)
count++;
cout << count;
```

Оператор % возвращает остаток от деления, оператор %= присваивает левому операнду остаток деления левого операнда на правый. Соответственно в результате выполнения выражения  $m \% = n$ ; целочисленная переменная  $m$  примет значение частного текущего значения этой переменной и  $n$ .

Рассмотрим следующую задачу.

В книге на одной странице помещается  $k$  строк. Таким образом, на 1-й странице печатаются строки с 1-й по  $k$ -ю, на второй — с  $(k+1)$ -й по  $(2k)$ -ю и т. д. Написать программу, определяющую по номеру строки в тексте номер страницы, на которой будет напечатана эта строка, и порядковый номер этой строки на странице.

Пользователь вводит:  $k$  — количество строк на странице и число  $n$  — номер строки в тексте ( $1 \leq k \leq 200$ ,  $1 \leq n \leq 20000$ ). Решение:

```
int k, n, page, line;
cin >> k >> n;
page = (n - 1) / k + 1;
line = (n - 1) % k + 1;
cout << page << ' ' << line;
```

## **Лабораторная работа № 2** **БАЗОВЫЕ ТИПЫ ЯЗЫКА C++**

**Цель работы** Получить представление о базовых типах данных языка программирования C++, изучить принципы выполнения операций на этих типах данных, познакомиться с приоритетом операций.

### **Постановка задачи**

1. Определить количество памяти, отводимое под переменные базовых типов.
2. Изучить способы приведения типов в языке C++.

3. Изучить возможность возникновения переполнения типов.
4. Изучить приоритет операций.

### ***Указания к выполнению работы***

1. Определить количество памяти, отводимое под переменные базовых типов.

– объявить переменные всех базовых типов языка C++, переменную типа `void` также необходимо объявить, убедиться в том, что объявление переменной типа `void` запрещено правилами языка программирования C++, закомментировать строку объявления соответствующей переменной, объявление не удалять;

– проинициализировать объявленные в предыдущем пункте переменные допустимыми для соответствующего типа значениями. Проинициализировать все переменные, за исключением переменной типа `void`;

– вывести на экран значения, объявленных и проинициализированных переменных, используя функцию `printf` со спецификатором, соответствующим типу данных выводимой переменной;

– определить количество памяти, отводимое под объявленные переменные базовых типов, используя `sizeof()`;

– используя результаты, полученные в предыдущем пункте рассчитать диапазон значений переменных типа `int` и `char`.

– расчёт производить, опираясь на информацию о количестве памяти занимаемой соответствующим типом данных и знания о принципах хранения данных.

2. Изучить способы приведения типов в языке C++.

а) Набрать следующий фрагмент кода:

```
int a = 65;
float b = 42.17;
char d = 'd';
a = b; printf("a =%i\n", a);
a = c; printf("a =%i\n", a);
a = 65; printf("a =%i\n", a);
b = a; printf("b =%f\n", b);
b = c; printf("b =%f\n", b);
b = 42,17; printf("b =%f\n", b);
```

```
c = a; printf("c =%c\n", c);
```

```
c = b; printf("c =%c\n", c);
```

б) Объяснить с точки зрения хранения данных значения, выведенные на экран.

в) Используя функцию `printf` вывести на экран переменные, указав спецификаторы, не соответствующие их типам данных. То есть, целочисленную переменную вывести со спецификатором для символьного и вещественного и типов, переменную вещественного типа данных со спецификаторами для целого и символьного типов, символьную переменную со спецификаторами для целого и вещественного типов данных.

г) Объяснить значения, выведенные на экран, основываясь на знаниях о хранении данных в памяти компьютера.

3. Изучить возможности возникновения переполнения типов переполнения типов.

а) Объявить две переменные символьного типа данных, проинициализировать их значениями 97 и 265, вывести значения этих переменных со спецификаторами для символьного типа данных, для целого знакового и беззнакового типов. Объяснить результаты, выведенные на экран.

б) Объявить переменную типа `signed char` и переменную типа `unsigned char`. Проинициализировать обе объявленные переменные значением 0. Вывести на экран значения переменных, используя функцию `printf` со спецификаторами для целого знакового и символьного типов. Изменить значение переменных, присвоив им число 255. Ещё раз вывести на экран значения, описанным выше способом. Увеличить значение каждой переменной на два и снова вывести на экран. Объяснить с точки зрения знаний о хранении данных все, полученные на экране значения.

в) Объявить переменную типа `int` и переменную типа `unsigned int`, проинициализировать объявленные переменные значениями, максимально допустимыми для соответствующих типов. Вывести значения переменных на экран используя функцию `printf` со спецификаторами, соответствующими типам переменных. Увеличить значение каждой переменной на три. Снова вывести на значение каждой переменной. Каждую переменную вывести со спецификатором для



знакового и беззнакового типов. Объяснить все значения, выведенные на экран.

#### 4. Приоритет операций.

Ниже представлены строки кода, которые нужно набрать. В некоторых выражениях допущены ошибки, которые необходимо исправить. Для исправления ошибок допустимо использовать только круглые скобки и пробел. Разбивать выражение на два не допускается. В отчёте указать порядок действий во всех выражениях. Вывести на экран значение искомых переменных и всех переменных значение, которых в выражении изменялось, как показано ниже. Объяснить результаты, выведенные на экран.

```
int d, i = 1;
d = ++i++;
std::cout << "d=" << d << " i=" << i << "\n";
d=0, i = 1;
d = ++i++ + (++d)++;
std::cout << "d=" << d << " i=" << i << "\n";
d = 1;
d += d++;
std::cout << "d=" << d << "\n";
d = 1;
d += ++d+2;
std::cout << "d=" << d << "\n";
int a, b, c, k;
b = 4; d = 7;
k = (a = b) + (c = d);
std::cout << "k=" << k << " a=" << a << " b=" <<
b << " d=" << d << " c=" << c << "\n";
int l, j; i = l = j = k = 0;
a = i++ && ++j || k || l++;
std::cout << "a=" << a << " i=" << i << " j=" << j
<< " k=" << k << " l=" << l << "\n";
a = 2; b = 1;
k = (a != b) ? (b++ - a) : (++b - a);
std::cout << "a=" << a << " b=" << b << " k=" << k << "\n";
a = 2; b = 3;
```

```
float y1, y2, f = 3.5;
y1 = f * a / b; y2 = f * (a / b);
std::cout<<"y1="<<y1<< " y2=" << y2 << "\n";
```

***Вопросы для самоконтроля, подготовки к выполнению и защите лабораторной работы***

1. Все типы данных в языке программирования можно разделить на две группы C++. Назовите эти группы.
2. Какие типы данных для хранения целочисленных значений существуют в языке программирования C++?
3. Расскажите о способе хранения целых беззнаковых переменных. Поясните графически.
4. Расскажите о способе хранения целых знаковых переменных. Поясните графически.
5. Расскажите о способе хранения вещественных переменных. Поясните графически.
6. Поясните, что называют переполнением типа данных?
7. Как будет определено значение переменной, при присвоении ей значения, выходящего за границу диапазона?
8. Какие типы данных используют для работы с вещественными переменными в языке программирования C++?
9. Назовите известные вам базовые типы языка C++.
10. В каких ситуациях используют приведение типов?
11. В каком случае приведение типов называют явным, в каком неявным?
12. Каким образом осуществляется приведение типов операндов в операторе присваивания.
13. Каким образом происходит работа с данными при вычислении выражений, содержащих разнотипные данные?
14. Какое значение будет получено при увеличении на единицу значения целочисленной переменной, хранящей максимально возможное значение для данного типа? Рассмотреть два случая: для знаковых и беззнаковых типов.

## Задания для самостоятельной работы

1. Напишите код, который выводит на консоль строку "Hello student!" и переводит курсор на следующую строку. Используйте функцию стандартной библиотеки C++.

2. Объявите последовательно следующие переменные:  $b$  – логическая переменная,  $c$  – символ,  $w$  – широкий символ,  $i$  – целое число,  $f$  – число с плавающей точкой,  $d$  – число с плавающей точкой двойной точности. Каждую переменную объявить в отдельной строке. Проинициализировать переменные, следующими значениями *true*, 'A', 655, 10, 1.1f, 1.21e22. Вывести значения переменных на экран в формате: *имя переменной=значение*;

3. Напишите простой калькулятор, который складывает два целых числа  $a$  и  $b$ . Результат записывается в переменную  $c$  и выводится на консоль. Пример результат работы программы:

Введите первое число: 3

Введите второе число: 3

Сумма чисел = 6

Найти и исправить ошибку в представленном ниже коде.

```
int main()
{
    int a = 15;
    int b = 2;
    float c = a/b;
    cout << c << endl;
    return 0;
}
```

4. Запрограммируйте вычисление следующего выражения:

$x = (a + b - f/a) + fa^2 - (a + b)$ . Числа  $a$ ,  $b$  и  $f$  вводятся с клавиатуры.

Результат вывести без объявления и использования переменной  $x$ , вычислив формулу сразу в выводе. `cout << "x = " <<...место для формулы...<<`

Вывод программы имеет следующий вид:

Введите числа  $a$ ,  $b$  и  $f$ :

$a = 15$

$b = 10$

$f = 3$

Результат вычисляем по формуле:  $x = (a + b - f/a) + faa - (a + b)$

$x = 675$

5. Вычислить и вывести на экран сумму цифр введенного с клавиатуры целого трёхзначного числа.

6. Найти число десятков целого неотрицательного числа, введенного с клавиатуры, то есть вторую справа цифру его десятичной записи.

7. Идёт  $k$ -я секунда суток. Определить, сколько целых часов  $h$  и целых минут  $m$  прошло с начала суток.

8. Определить сколько сейчас целых часов  $h$  и целых минут  $m$ . Если известно, что часовая стрелка повернулась с начала суток на  $d$  градусов. Пользователь вводит  $d$ , такое, что  $0 \leq d < 360$ .

9. Даны значения двух моментов времени, принадлежащих одним и тем же суткам: часы, минуты секунды для каждого из моментов времени. Известно, что второй момент времени наступил не раньше первого. Определить, сколько секунд прошло между двумя моментами времени. В первой строке входных данных находятся три целых числа — часы, минуты и секунды первого момента времени. Во второй строке — три числа, характеризующие второй момент времени. Число часов лежит в диапазоне от 0 до 23, число минут и секунд — от 0 до 59.

10. Вычислить общую сумму покупки нескольких товаров. Например, плитки шоколада, кофе и пакета молока.

11. Объявить три переменные типа *int* и присвоить первой числовое значение, вторая переменная равна первой переменной, увеличенной на 3, а третья переменная равна сумме первых двух.

12. Подсчитать общее количество предметов для сервировки стола. Например, чашки, такое же количество блюдец и ложек.

13. Написать программу для перевода сантиметров в дюймы. Для справки: в одном дюйме 2.54 сантиметра.

14. Код представленный ниже вычисляет сумму первых  $N$  натуральных чисел по формуле суммы арифметической прогрессии:

```
int n;  
cin >> n;  
cout << n * (n + 1) / 2 << "\n";
```

Программа должна работать для всех  $N \leq 4000000000$ , но для некоторых  $N$  она работает неправильно. Найти причину проблемы и исправить программу.

### 3. ОПЕРАТОРЫ ВЕТВЛЕНИЯ ЯЗЫКА C++

#### *Условный оператор if*

Оператор `if` позволяет реализовать алгоритмическую конструкцию ветвление на языке программирования C++. Оператор имеет полную и сокращённую форму. Последняя используется при отсутствии действий на ветке «иначе», т.е. при невыполнении условия. Синтаксис оператора такой:

`if (выражение) оператор1; [else оператор 2;]`

Выполнение оператора начинается с вычисления выражения. Выражение может быть любым: арифметическим, логическим или сложным. В любом случае результат вычисления выражения, будет интерпретирован как логическое значение. При этом, ноль – «ложь», любое отличное от нуля значение – «истина».

Если результат вычисления выражения – «истина», выполняется оператор1. В противном случае выполняется оператор2, если это полная форма оператора `if`, или ничего если используется не полная форма оператора ветвления и оператор2 отсутствует. Оператором1 и оператором2 могут быть любые операторы языка программирования C++, если на месте одного из операторов нужно выполнить несколько действий, используют составной оператор. Т.е. заключают эти действия в фигурные скобки. Таким образом не нарушаются правила использования оператора `if`. Рассмотрим несколько примеров.

Полная форма оператора `if`.

```
int c = 7, d = 9, max = 0;
if (c > d) max = c; else max = d;
```

Неполная форма оператора `if`.

```
if (c > d) max = c;
```

Использование составного оператора.

```
if (c < d && (c > d || c == 0)) d++; else { d*= c; c++;
}
```

Так как оператор1 и оператор2 могут быть любыми, значит на их месте может быть использован оператор `if`. Такое ветвление называют вложенным, Например:

```
int c = 6, d = 7, f = 9, max = 0;
if (c > d) { if (c > f) max = c; else max = f; }
    else if (d > f) max = d; else max=f;
cout << "max="<<max;
```

### ***Оператор множественного ветвления switch***

В ситуациях, когда выражение является целочисленным и возможно более чем два варианта ответа, целесообразнее использовать оператор множественного ветвления `switch`.

Правило использования оператора следующее:

```
switch (<целочисленное выражение>)
{
case константное выражение1: оператор1;
case константное выражение2: оператор2;
case константное выражение3: оператор3;
...
case константное выражениеN-1: операторN-1;

[default: <оператор N>]
}
```

После вычисления выражения, происходит последовательное сравнение его значения со значениями констант. Все константы, используемые в операторе, должны быть уникальными, т.е. не должно быть повторений. Сравнение происходит до первого совпадения. Если совпадение найдено, происходит последовательное выполнение всех операторов, без проверки на совпадение, включая оператор, указанный после ключевого слова `default`. Если, необходимо выполнить только действия, относящиеся к определённой константе, следует использовать составной оператор. Последним действием этого оператора, должен быть оператор `break`, передающий управление оператору, следующему за оператором `switch`.

Если ни одна константа не совпала со значением выражения, будет выполнен оператор, указанный после ключевого слова

default. В случае, когда при отсутствии совпадений, никаких действий выполнять не нужно, default можно опустить.

Простые примеры использования оператора switch приведены ниже.

```
int i = 1;
switch (++i)
{
case 1:printf("1\n");
case 2:printf("1+1=2\n");
case 3:printf("2+1=3\n");
case 4:printf("3+1=4\n");
default:printf("No!\n");
}
```

Результат выполнения этого фрагмента кода будет таким:

```
1+1=2
2+1=3
3+1=4
No!
```

Внесём изменения в оператор switch, добавим оператор break.

```
int i = 1;
switch (++i)
{
case 1: {printf("1\n"); break; }
case 2: {printf("1+1=2\n"); break; }
case 3: {printf("2+1=3\n"); break; }
case 4: {printf("3+1=4\n"); break; }
default:printf("No!\n");
}
```

В результате выполнения этого фрагмента на экране появится следующая строка: 1+1=2.

Если изменить значение переменной *i*. И задать его меньше нуля либо больше четырёх, на экране появится строка «No!».

### ***Тернарная операция***

Тернарная операция имеет три операнда. К ней прибегают, когда возникает необходимость не только реализовать ветвление, но и

вернуть некоторое значение в точку вызова. Синтаксис операции такой:

```
(<операнд1>)?<операнд2>:<операнд3>;
```

Операнд1 - выражение, результат которого интерпретируется логически. В зависимости от его значения будет выполнен операнд2 или операнд3. Если результат вычисления выражения – «истина», будет выполнен операнд 2, иначе - операнд 3.

Операнд2 и операнд3 представляют собой операторы. Результат выполнения соответствующего оператора и будет результатом тернарной операции, это значение будет возвращено в точку вызова.

Рассмотрим примеры применения тернарной операции.

```
float q = 2.4;
float w = -1.7;
float f = (q < w) ? q++ : w++;
cout << " f =" << f << endl;
cout << "w=" << w << endl;
cout << "q=" << q;
```

Этот фрагмент кода будет выполнен следующим образом. В первых двух строках объявлены и проинициализированы переменные w и q. В третьей строке объявлена переменная f, значение которой будет определено возвращаемым значением тернарной операции. Разберём подробнее выполнение тернарной операции. Значение первого операнда – «ложь», т.к. значение переменной q не меньше значения переменной w. Таким образом следующим будет выполнен операнд3, постфиксный инкремент переменной w. Соответственно, в точку вызова будет возвращено и сохранено в переменной f начальное значение переменной w, после чего значение переменной будет увеличено на единицу. Результат выполнения этого фрагмента кода будет такой:

```
f=-1.7
w=-0.7
q=2.4
```

Разберём менее тривиальный пример.

```
int m=0, i, j, k; i = 9; j = 15;
k=i<j?i++:++j?m=i+j, printf("i=%i\n", i) : i;
cout<<"i="<<i<<endl;
cout<<"j="<<j<<endl;
```



```
cout<<"m="<<m<<endl;  
cout<<"k="<<k<<endl;
```

В результате выполнения на экран будут выведены следующие значения:

```
i = 10  
j = 15  
m = 0  
k = 9
```

Поясним как получились такие значения. Первый операнд имеет значение –«истина», следовательно, выполняется второй операнд `i++`, возвращаемое значение – 9, будет сохранено в переменной `k`. Третий операнд, который представляет собой тернарную операцию, выполнен не будет, поэтому значения переменных `j` и `m` не будут изменены.

Изменим начальные значения переменных `i` и `j` так, чтобы проверка условия возвращала – «ложь». Например, так: `int i = 19; j = 5`; остальную часть кода оставим без изменений. Тогда после проверки условия будет выполнен третий операнд. Рассмотрим его подробнее. Как мы знаем на месте третьего операнда может быть оператор, в данном случае это тернарная операция, первый операнд, которой значение префиксного инкремента переменной `j`. Начальное значение переменной `j` – 5, в точку вызова возвращается измененное значение - 6, которое трактуется как истина. Таким образом будут выполнены операторы, входящие в состав второго операнда `m=i+j`, `printf("i=%i\n", i)`. Переменная `m` станет равной 25, результат сложения начального значения переменной `i` и увеличенного на единицу значения переменной `j`. Оператор `printf("i=%i\n", i)` выводит на экран строку «`i=19`» и возвращает в точку вызова количество выведенных символов, включая символ «`\n`». Это значение записывается в переменную `k`, таким образом значение переменной `k` становится равным 5. Результат на экране будет следующим:

```
i=19  
i = 19  
j = 6  
m = 25  
k = 5
```

## Лабораторная работа № 3 ОПЕРАТОРЫ ВЕТВЛЕНИЯ В ЯЗЫКЕ C++

### ***Цель работы***

Повторить алгоритмическую конструкцию ветвление. Изучить синтаксис оператора ветвления *if*, оператора множественного ветвления *switch* в языке программирования C++, тернарной операции. Получить навыки графического представления разветвлённых алгоритмов и их реализации в виде программного кода на языке программирования C++.

### ***Постановка задачи***

1. Пользователь вводит параметр  $x$ . Вычислить значение переменной  $y$ , зависящей от  $x$ :

$$y = \begin{cases} x - 12, & x > 0 \\ 5 \cdot x - 0 & \\ x^2, & x < 0 \end{cases}$$

2. Ввести с клавиатуры целое число. Интерпретируя это число как возраст мужчины, заполняющего анкету, вывести на экран одно из четырёх сообщений:

– если указан возраст от 18 и до 27 лет, сообщать, что человек подлежит призыву на срочную службу или может служить по контракту;

– если указан возраст от 28 до 59 лет, сообщать, что заполняющий анкету может служить по контракту.

– если указан возраст менее 18 или более 59 лет, сообщить о том, что заполняющий находится в непризывном возрасте.

– если указан неположительный возраст, сообщение об ошибке.

3. Ввести номер дня недели. Вывести на экран расписание занятий на этот день.

Реализовать два варианта.

3.1 Вводимое значение должно быть целым числом в диапазоне от одного до семи, иначе вывести сообщение об ошибке.

3.2 Пользователь вводит номер дня в семестре и день недели с которого семестр начался.

Вводимое значение может быть любым целым числом.

Выдать сообщение об ошибке, если число меньше единицы.

Выдать сообщение о превышении диапазона, если число больше количества дней в семестре или день с которого начался семестр меньше 1 или больше 7.

Во всех остальных случаях выдать название дня недели и расписание на этот день.

4. Ввести положительное число. Вывести на экран название животного – символ года по восточному календарю.

### ***Вопросы для самоконтроля, подготовки к выполнению и защите лабораторной работы***

1. Запишите синтаксис и приведите пример использования условного оператора `if`.

2. В каком случае используют полную форму условного оператора `if`?

3. В каком случае используют сокращённую форму условного оператора `if`?

4. Запишите синтаксис и пример использования оператора множественного ветвления `switch case`.

5. В какой ситуации предпочтительнее использовать оператор `if`, а в какой оператор `switch`?

6. В каком случае будет выполнен оператор, указанный после ключевого слова `default` в операторе множественного выбора?

7. В каком случае ключевое слово `default` в операторе множественного выбора можно опустить?

8. Каким образом осуществляют досрочный выход из оператора `switch`?

9. Запишите синтаксис и приведите пример использования тернарного оператора.

10. Что является возвращаемым значением тернарного оператора?

### *Задания для самостоятельной работы*

1. Если погода будет хорошая, школьник пойдёт гулять, если плохая, то будет делать уроки. Ввести информацию о погоде, вывести сообщение о прогулке.

2. При покупке товара на сумму более 1000 рублей предоставляется скидка 15%. Вычислить сумму покупки. Цену товара вводит пользователь.

3. Найти наибольшее целое число из трех введенных с клавиатуры.

4. Определить является ли введенное пользователем число чётным.

5. Проверить делится ли введенное число на три.

6. Определить лежит ли точка с указанными координатами в круге радиуса  $R$  с центром в начале координат. Значение радиус вводит пользователь.

7. Ракета запускается с Земли со скоростью  $V$ (км/час) в направлении движения Земли по орбите вокруг Солнца. Определить результат запуска космической ракеты с Земли в зависимости от скорости  $V$ . Известно, что при  $V < 7,8$  ракета упадет на Землю; при  $7,8 < V < 11,2$  ракета станет спутником Земли; при  $11,2 < V < 16,4$  ракета станет спутником Солнца; при  $V > 16,4$  ракета покинет солнечную систему. Значение скорости вводит пользователь.

8. Пользователь вводит длину стороны квадрата и радиус круга. Определить какое из утверждение верное: «Круг может быть вписан в квадрат» или «Квадрат может быть вписан в круг».

9. Написать программу, анализирующую возраст человека и относящая его к одной из четырех групп: дошкольник, ученик, работающий, пенсионер. Возраст человека вводится с клавиатуры.

10. Написать программу, которая по введенному  $K$  – числу грибов выводит на экран фразу «Мы нашли в лесу  $K$  грибов», причем согласовывает окончание слова «гриб» с числом  $K$ . Количество грибов может быть любым целым числом. Окончание фразы определяется значением последней цифры.

## 4. ОПЕРАТОРЫ ОРГАНИЗАЦИИ ЦИКЛОВ ЯЗЫКА C++

Операторы циклов позволяют реализовать алгоритмическую конструкцию «повторение».

Независимо от того каким оператором будет реализован цикл, необходимо правильно организовать следующие три этапа работы с ним: задание начальных значений и/или параметров цикла, тело цикла, условие продолжения цикла.

*Телом цикла* называют многократно повторяющиеся действия.

*Итерацией* называют однократное выполнение тела цикла.

Условие в зависимости от значения которого будет выполнена следующая итерация или осуществлён выход из цикла называют *условием продолжения цикла*.

В зависимости от взаимного расположения условия продолжения цикла и тела цикла различают цикл с *предусловием* (условие расположено перед телом цикла) и цикл с *постусловием* (условие расположено после тела цикла).

На рис. 16 представлены структурные схемы цикла с предусловием и пост условием.

Если из условия задачи ясно, что тело цикла необходимо выполнить как минимум один раз, используют цикл с постусловием, если же возможна ситуация, в которой тело цикла не выполнится ни единого раза, используют цикл с предусловием.

*Параметрами* цикла называют переменные, используемые в условии продолжения цикла. *Счётчиком* цикла называют целочисленные параметры цикла, изменяющиеся с определённым шагом.

Для того чтобы цикл работал корректно, в его теле обязательно должно быть предусмотрено изменение параметров цикла. Начальное значение всех обрабатываемых в цикле переменных, включая параметры цикла должно быть задано до тела цикла.

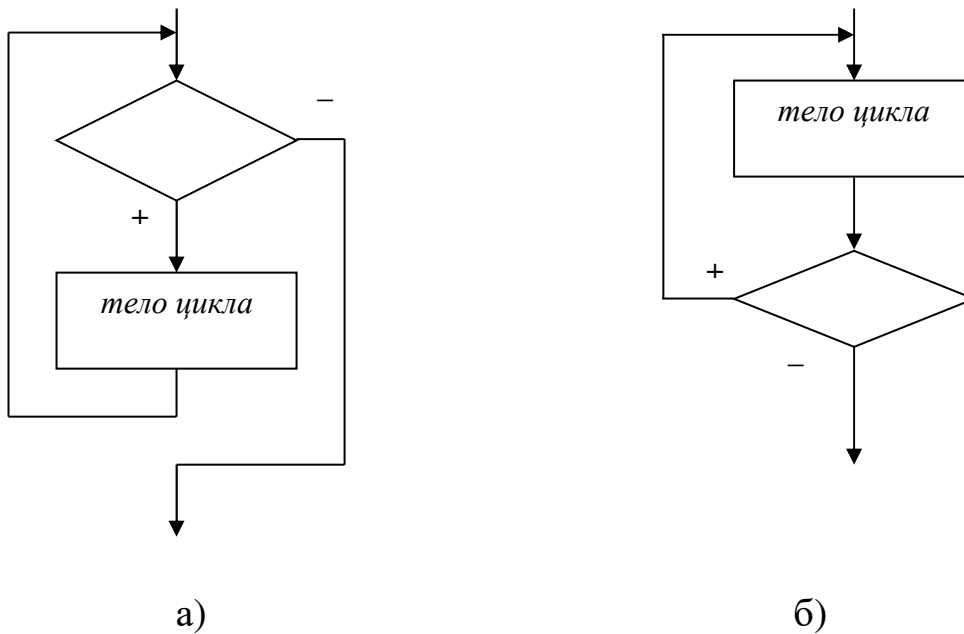


Рис. 16. Структурная схема организации цикла  
 а - цикл с предусловием; б - цикл с постусловием

### ***Цикл с предусловием while***

Оператор `while` позволяет реализовать цикл с предусловием. Синтаксис оператора такой:

`while (выражение) оператор;`

Если результат вычисления выражения – «истина», будет выполнено тело цикла. Если результат вычисления выражения – «ложь» управление будет передано оператору, следующему за телом цикла. В случаях, когда тело цикла должно содержать более одного оператора используют фигурные скобки, объединяющие последовательность операторов в один оператор, называемый составным оператором. Таким образом, требование к тому что бы после выражения стоял оператор – тело цикла, не нарушается.

Разберём простой пример, использования оператора `while` для вычисления произведения пяти введенных с клавиатуры чисел.

```
int p = 1;
int const n = 5;
int a = 0, i = 1;
while (i <= n)
{   printf("a=");
    scanf_s("%i", &a);
```

```

    p = p * a; i++;
}
printf("p=%i\n", p);

```

На рис. 17 представлен результат работы программы.

```

Консоль отладки
a=1
a=2
a=3
a=4
a=5
p=120

```

Рис. 17. Результат вычисления произведения пяти значений с использованием цикла *while*

### ***Цикл с пост условием do while()***

Оператор `do..while` позволяет реализовать цикл с постусловием. Использовать этот оператор следует, в ситуациях, когда из условия задачи понятно, что тело цикла необходимо выполнить как минимум один раз. Синтаксис оператора следующий:

```
do <оператор> while (выражение);
```

После выполнения оператора – тела цикла, вычисляется выражение. В зависимости от результата вычисления выражения будет осуществлён переход на следующую итерацию цикла, если результат – «истина», или выход из цикла, результат – ложь.

В качестве примера вычислим произведение пяти вводимых с клавиатуры элементов с помощью оператора `do while`. Результат работы приведённого примера, представлен на рис. 18.

```

int const n = 5;
int p = 1, a = 0, i = 1;
do {printf("a=");
    scanf_s("%i", &a);
    p = p * a; i++;
} while (i < n);
printf("p=%i\n", p);

```

```

Консоль отладки
a=3
a=5
a=6
a=7
p=630

```

Рис. 18. Результат работы фрагмента программы вычисления произведения элементов с использованием цикла *do while*

Легко заметить, что во втором количество итераций меньше, при одинаковых начальных значениях примере меньше.

### ***Цикл с параметром for***

Оператор `for` позволяет реализовать цикл с предусловием и условием продолжения.

Правило записи оператора такое:

```
for (выражение1; выражение2; выражение3;)оператор;
```

Выражение1 называют *инициализатором*. Как правило инициализатор содержит объявления и инициализацию переменных – параметров цикла. Инициализатор выполняется один раз до начала выполнения цикла. Инициализатор может содержать несколько операторов, разделённых запятой.

Выражение2 – условие продолжения цикла. Если результат вычисления выражения – «истина», выполняется тело цикла, в противном случае - управление передаётся оператору, следующему за оператором `for`.

Выражение3 называют *итератором*, обычно содержит оператор, изменяющий значение параметра цикла. Выполняется этот оператор после выполнения всех операторов тела цикла. Далее снова осуществляется проверка условия - выражения2.

Если тело цикла содержит, более одного оператора, его заключают в фигурные скобки.

Вычислим произведение пяти введенных с клавиатуры чисел с использованием оператора `for`. На рис. 19 представлен результат выполнения приведённого примера

```
int const n = 5;
int p = 1, a = 0, i = 1;
for (int i = 0; i < n; i++)
{printf("a=");
scanf_s("%i", &a);
p*=a; }
printf("p=%i\n", p);
```

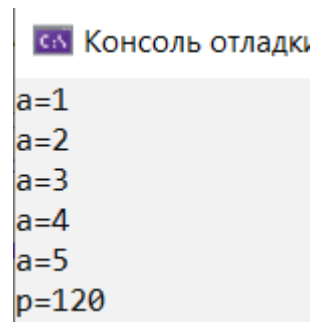


Рис. 19. Результат работы фрагмента программы вычисления произведения элементов с использованием цикла `for`

Любое из выражений, указанных в круглых скобках после ключевого слова `for` может быть пропущен. Возможна ситуация, в которой все три выражения отсутствуют. Какое бы из выражений не



было опущено наличие точки с запятой, завещающей это выражение обязательно.

Если отсутствует первое выражение, инициализация параметра цикла должна быть выполнена до ключевого слова `for`. Если отсутствует второе выражение, выход из цикла необходимо предусмотреть в теле цикла. Обычно для этого используют оператор `break`, передающий управление оператору, следующему сразу за циклом. Если отсутствует третье выражение, необходимо включить итератор в тело цикла.

Внесём изменения в рассмотренный выше пример, опуская поочередно выражения в круглых скобках, после ключевого слова `for`.

Удалим из круглых скобок инициализатор и разместим его до цикла.

```
int p = 1; int a = 0;
int const n = 5;
int i = 0; // инициализация параметра цикла i
for (; i < n; i++)
{
    scanf_s("%i", &a);
    p = p*a;
}
```

Здесь важно понимать, что при таком подходе переменная *i* существует во всём блоке, в котором содержится оператор `for`. В предыдущем же примере переменная *i* существовала только в теле цикла `for`.

Разберём пример, в котором в круглых скобках отсутствует третье выражение. Как было сказано ранее, в таком случае итератор необходимо разместить последним оператором тела цикла.

```
int p = 1;
int const n = 5;
int a = 0;
for (int i = 0; i < n; )
{
    printf("a=");
    scanf_s("%i", &a);
    p = p * a;
}
```

```

        i++;
    }
    printf("s=%i\n", s);

```

Обратите внимание, на то, что, не смотря на отсутствие определённых выражений в двух приведённых выше примерах, точка с запятой стоит на своём месте.

Нам осталось рассмотреть пример использования цикла `for` с отсутствием второго выражения.

```

int p = 1;
int const n = 5;
int a = 0;
for (int i = 0; ;i++)
{
    if (i>=n)break;
    printf("a=");
    scanf_s("%i", &a);
    p = p * a;
}
printf("s=%i\n", s);

```

Так как условие продолжения цикла отсутствует, в теле цикла использован оператор `if`, позволяющий проанализировать значение счётчика и осуществить вход из цикла.

Теперь рассмотрим самую интересную ситуацию, отсутствие в круглых скобках всех выражений. Не забудем про необходимость точек с запятой `for(;;)`.

```

int p = 1;
int const n = 5;
int a= 0;
int i = 1;
for(;;)
{if (i>n) break;
    printf("a=");
    scanf_s("%i",&a);
    p = p * a; i++;
}
printf("s=%i\n", s);

```

Если в любом из рассмотренных операторов цикла необходимо выполнить досрочный выход используют операторы: `break` и `return`.

Как только оператор `break` будет встречен в теле цикла, управление будет передано оператору, следующему за циклом. Операторы, стоящие в теле после ключевого слова `break`, выполнены не будут. После того как оператор `break` будет встречен в коде, все последующие операторы тела цикла игнорируются и управление передаётся оператору, следующему за оператором в теле, которого встретился оператор выхода `break`.

Разберём пример с использованием в теле цикла оператора `break`. Будем вычислять сумму пяти вводимых с клавиатуры чисел, в случае на  $i$ -ом шаге сумма превысит некоторое значение  $N$ , будет осуществлён досрочный выход из цикла.

```
int s = 0;
int const n = 5;
int const N = 15;
int a = 0;
int i = 1;
while (i <= n)
{
    printf("a=");
    scanf_s("%i",&a);
    s = s + a; i++;
    if (s > N) break;
}
printf("s=%i\n", s);
```

На рис. 20 представлены результаты работы приведенного выше фрагмента кода с различными входными значениями.

```
Консоль отладки Microsoft Visual Studio
a=2
a=5
a=7
a=8
s=22
```

а)

```
Консоль отладки Microsoft Visual Studio
a=0
a=2
a=0
a=1
a=3
s=6
```

б)

Рис. 20. Результат работы программы вычисления суммы с использованием оператора досрочного выхода из цикла: а) условие досрочного прерывания выполнилось; б) досрочный выход не произошёл, выполнились все итерации цикла

Использование оператора `return` приводит к досрочному выходу из функции в теле которой он был использован.

Использование в теле цикла оператора `continue` позволяет перейти на следующую итерацию, минуя оставшиеся операторы тела цикла.

```
for (int count = 0; count <= 30; count++)
    {if ((count % 5) != 0) continue;
        std::cout << count << std::endl; /* Это действие будет пропущено, если число не делится нацело на 5*/
    }
```

В результате выполнения этого цикла на экран будет выведен столбец чисел от нуля до тридцати, делящихся нацело на пять.

## Лабораторная работа № 4

### ОПЕРАТОРЫ ОРГАНИЗАЦИИ ЦИКЛОВ ЯЗЫКА C++

#### **Цель работы**

Повторить алгоритмическую конструкцию – цикл. Изучить синтаксис операторов организации циклов языка программирования C++, получить навыки разработки циклических алгоритмов, их графического представления и реализации на языке программирования C++.

#### **Постановка задачи**

1. Сколько всего километров пробежит спортсмен за  $n$  дней. Если в первый день он пробежал  $s$ , километров, а каждый следующий день пробегал на 15% больше чем в предыдущий.

2. Каждые 3 часа амеба делится на 2 клетки. Определить сколько амёб будет через  $k$  часов. Начальное количество амёб и  $k$  - количество прошедших часов вводит пользователь.

3. Если известно, что у уток и зайцев вместе 64 лапы. Сколько может быть зайцев и уток. Вывести на экран все возможные варианты.

Для решения любой задачи лабораторной работы допустимо использовать любой из операторов `while`, `do...while`, `for`. Обязательное условие: каждый из этих операторов в лабораторной работе должен быть использован, хотя бы один раз.

### ***Вопросы для самоконтроля, подготовки к выполнению и защите лабораторной работы***

1. Что называют в программировании циклом?
2. Что называют телом цикла?
3. Какой цикл называют циклом с предусловием?
4. Какой цикл называют циклом с постусловием?
5. Что называют итерацией?
6. Какие операторы организации цикла в языке программирования C++ вы знаете?
7. Оператор `for` позволяет организовать цикл с предусловием или постусловием? Напишите синтаксис и приведите пример использования этого оператора.
8. Оператор `while` позволяет организовать цикл с предусловием или постусловием? Напишите синтаксис и приведите пример использования этого оператора.
9. Оператор `do...while` позволяет организовать цикл с предусловием или постусловием? Напишите синтаксис и приведите пример использования этого оператора.
10. В каком случае говорят о досрочном выходе из цикла?
11. Какой оператор нужно использовать для досрочного выхода из цикла?
12. Какой оператор нужно использовать для прерывания текущей и перехода на следующую итерацию цикла?

### *Задания для самостоятельной работы*

1. Вывести на экран квадраты всех натуральных чисел до  $N$  включительно.  $N$  вводит пользователь.
2. Вывести на экран кубы чисел от  $A$  до  $B$  включительно.  $A$  и  $B$  вводит пользователь.
3. Написать программу, которая возводит число в целочисленную степень. Число и степень ввести с клавиатуры.
4. Найти наибольшее из  $N$  введённых с клавиатуры чисел.
5. Найти наименьшее из  $N$  введённых с клавиатуры чисел.
6. Найти сумму  $N$  введённых с клавиатуры чисел.
7. Найти среднее арифметическое  $N$  введённых с клавиатуры чисел.
8. Составить таблицу значений функции  $y = 5 - x^2/2$  на отрезке  $[-5; 5]$  с шагом 0.5.
9. Вычислить факториал числа, которое вводит пользователь.
10. Вывести на экран ряд чисел Фибоначчи, состоящий из  $N$  элементов. Значение  $33$  вводится с клавиатуры.  
Числа Фибоначчи – это последовательность чисел, начинающаяся с нуля и единицы, каждое последующее число равно сумме двух предыдущих, 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144,
11. Написать программу, возводящую в квадрат число, введённое пользователем. Предложить пользователю выбор: ввести новое число или выйти из программы.
12. Пользователь вводит целое положительное число. Вывести цифры числа в обратном порядке. Т.е. если ввели 123, вывести 321.
13. Пользователь вводит целое неотрицательное число. Найти максимальную цифру числа.
14. Пользователь вводит целое неотрицательное число. Найти минимальную цифру числа.
15. Пользователь вводит целое неотрицательное число. Найти сумму цифр числа.
16. Пользователь вводит целое неотрицательное число и цифру. Удалить эту цифру из числа.
17. Пользователь вводит целое неотрицательное число, найти сумму первой и последней цифр числа.
18. Вывести все делители целого неотрицательного числа, которое вводит пользователь.

19. Определить сколько можно приобрести ручек (по цене 20 руб.), карандашей (по 15 руб.) и ластиков (по 12 руб.) на 1000 рублей. При этом всего предметов должно быть 30.

20. Пользователь вводит два числа  $x$  и  $y$  и знак арифметической операции (+, -, /, \*). Написать программу, которая производит вычисления, вычисляет результат и запрашивает у пользователя дальнейший сценарий: «выйти или продолжить». В случае выбора - «продолжить» осуществить повторный ввод данных.

## 5. УКАЗАТЕЛИ И ССЫЛКИ. МАССИВЫ. ДИНАМИЧЕСКОЕ РАСПРЕДЕЛЕНИЕ ПАМЯТИ

### 5.1. УКАЗАТЕЛИ И ССЫЛКИ

*Указатель* – производный тип данных, хранящий адрес какой-либо ячейки памяти. Тип указатель должен быть связан с типом указуемого. Т.е. нельзя объявить просто указатель, можно объявить только указатель на данные определённого типа.

Для объявления указателя необходимо перед его именем написать тип данных указуемого и символ \*. Синтаксис:

*<тип данных>\*<имя указателя>;*

Например: `int *p;`  $p$  – указатель на целое число.

Так как указатель содержит адрес ячейки памяти, при его инициализации необходимо в правой части оператора присваивания указать адрес конкретной области данных.

Если указатель настраивают на переменную или константу, то перед соответствующим идентификатором ставят символ амперсанд &, означающий операцию взятия адреса.

```
int w =5;
```

```
int* p;
```

```
p=&w;
```

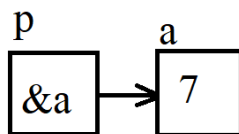
Инициализация указателя можно совместить с объявлением.

Например:

```
int a =7;
```

```
int* p=&a;
```

Результат работы написанных выше строк кода можно представить схематично, рис.21.



*Рис. 21. Схематичное представление связи с указателя с указуемым*

Стрелка на рис. 20 означает, что указатель *p* содержит адрес переменной *a*, иначе говоря, указывает на переменную *a*.

Через переменную типа указатель возможна работа как с самим значением указателя, так и со значением указуемого.

Для работы с указуемым, необходимо совершить переход по адресу. Для этого используют операцию разадресации (разыменовывание). Чтобы разыменовать указатель, т.е. перейти по адресу к указуемому, нужно использовать символ *\** перед уже объявленным и проинициализированным указателем.

Например, строка `*p=5;` означает запись в указуемое - переменную *a* значение 5.

Возможно объявление константного указателя, т.е. указателя значение, которого нельзя изменить. Для объявления такого указателя необходимо перед его именем указать ключевое слово `const`.

```
int a=7;
int * const p =&a;
```

Если `const` указать перед звездочкой, то будет объявлен не константный указатели, а указатель на константу.

```
const int a=10;
int const *p =&b;
const int c = 30;
p = &c;
printf("*p=%d", *p); // *p=30
```



Типом узазуемого может быть и указатель, т.е. возможно объявление указателя на указатель.

```
int a = 7;
int* p = &a;
int** pp = &p;
**pp = 9;
```

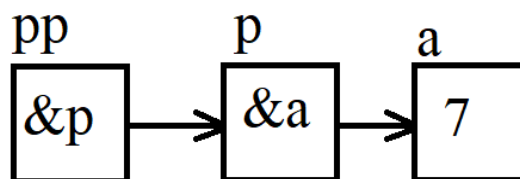


Рис. 22. Схематичное представление инициализации цепочки указателей

На рис. 22. схематично представлено результаты работы написанных выше строк кода.

Переменная `pp` – имеет тип указатель на указатель на целое. После выполнения последней строки кода, значение переменной `a`, будет равно 9.

Мы разобрали объявление, инициализацию и разыменовывание указателей. На указателях определены следующие операции: сравнения, вычитания, сложения с константой, инкремента и декремента.

Наглядно эти операции можно пояснить, настраивая указатели на элементы массива. Под массивом понимают поименованную последовательность элементов, расположенных в памяти подряд.

`int mas[5] = {1,2,3,4,5};` - объявлен и проинициализирован массив из пяти целых чисел. Нумерация элементов в массиве начинается с нуля.

Настроить указатель на элементы массива можно так:

```
int *p1 = &mas[0]; // указатель настроен на нулевой элемент
int* p2 = &mas[1]; // указатель настроен на первый элемент
```

Имя массива с точки зрения языка рассматривается как константный указатель, тогда указатели настроить можно следующим образом:

```
int *p1 = mas;
int* p2 = mas+1;
```

Указатели `p1` и `p2` настроены на нулевой и первый элементы массива соответственно. Результат этой операции схематично показан на рис. 23.

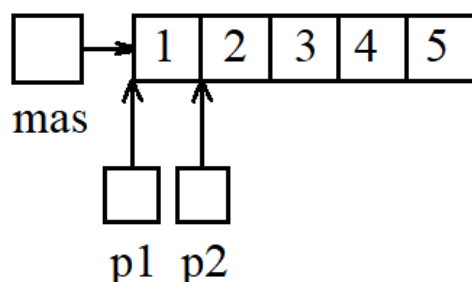


Рис. 23. Схематичное представление настройки указателей на элементы массива

При сравнении указателей сравниваться будут адреса переменных, на которые указатели настроены. Например:

```
if (p1 < p2)printf("p1\n");else printf("p2\n");
```

В данном случае (`p1 < p2`) – истина, т.к. указатели настроены на последовательно расположенные элементы, адрес указуемого `p2` старше.

Разберём ещё несколько примеров.

`int* p3 = p2;` - указатель `p3` будет настроен туда же, куда настроен указатель `p2`.

`p3++;` - значение `p3` увеличится на один размер типа указуемого.

`(*p3)++;` - увеличение на единицу значения указуемого.

На рис.24 схематично представлены приведённые примеры.

При работе с указателями на различные типы данных используют операцию приведения типов.

Операции постфиксного и префиксного инкремента и декремента изменяют значение указателя на единицу. За единицу принимается размер типа указуемого. Таким образом указатель будет перенастроен.

При вычитании указателей получаем целое число – количество ячеек указуемого типа между указуемыми.

```
int razn = p3-p1;
```

Переменная `razn` получит значение 2, что понятно из рис. 24.

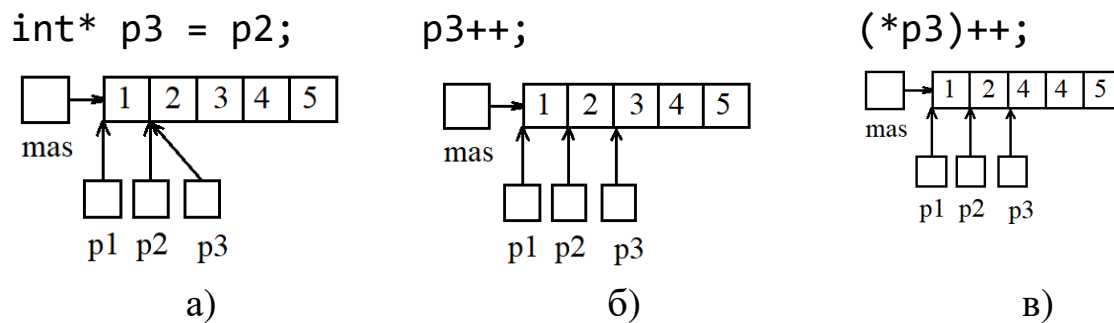


Рис. 24. Схематичное представление операций на указателях: а - настройка указателя `p3` на ячейку `mas[2]`, б - сдвиг `p3` на одну ячейку размера указуемого, в - изменение значения указуемого

Ссылка также как указатель содержит адрес переменной, с которой связана. Инициализация ссылки обязательна при объявлении.

`<тип данных> & <имя ссылки> = <переменная>;`

Наибольшее применение ссылки находят при передаче параметров в функции. Удобство ссылки в том, что для обращения к переменной с которой она связана не нужно производить дополнительные операции. При использовании ссылки операции будут производиться с переменной, с которой она связана. Перенастроить ссылку нельзя.

```
float f=10.3;
float h=12.34;
int &ref=f;//объявление ссылки ref и связь её с переменной.
ref = h; /*переменной f присвоено значение переменной h,
ссылка ref по прежнему связана с переменной f*/
ref++;//изменение значения переменной, связанной с ref
```

### ***Вопросы для самоконтроля, подготовки к выполнению и защите лабораторной работы***

1. Что в программировании называют указателем?
2. Для каких целей используют указатели?
3. Что называют указуемым?
4. Какие действия нужно выполнить, чтобы изменить значение указуемого через указатель?

5. Возможно ли объявление константного указателя?
6. Возможно ли объявление указателя на константу?
7. Возможно ли перенастройка указателя на константу?
8. Возможно ли объявить и настроить указатель на указатель?
9. Перечислите операции, применимые к указателям?
10. Что в программировании называют ссылкой?
11. Для каких целей использует ссылки?
12. Возможно ли перенастроить ссылку?
13. Возможно ли изменить значение переменной, связанной со ссылкой?
14. Чем с точки зрения языка программирования C++ является имя массива?
15. Продемонстрируйте работу с элементами массива через константный указатель, настроенный на первый элемент.

### ***Задания для самостоятельной работы***

1. Объявить и проинициализировать целочисленную переменную. Объявить указатель на целое и настроить на эту переменную. Обращаясь к указуемому через указатель увеличить значение первого втрое и вывести на экран.
2. Объявить и проинициализировать целочисленную переменную. Объявить указатель на целое и настроить на эту переменную. Обращаясь к указуемому через указатель вывести на экран увеличенное вдвое значение указуемого не изменяя его.
3. Вычислить значение потенциальной энергии  $E_p = mgh$ . Для этого объявить и проинициализировать вводом с клавиатуры соответствующие переменные ( $m$  и  $h$ ) и указатели, настроить указатели на переменные. Объявить вещественную константу задать её значение равным приближённое значение ускорения свободного падения. Связать с этой константой указатель соответствующего типа. Вычисление потенциальной энергии производить, обращаясь к множителям через соответствующие указатели.
4. Объявить и проинициализировать вещественную переменную  $k$ . Изменить и вывести на экран значение этой переменной, обращаясь к ней через указатель  $cp$ . При этом  $cp$  - константный указатель на указатель на вещественную переменную. Выполнить необхо-

димое объявление и инициализацию дополнительного указателя, таким образом, чтобы обращение к  $k$  через  $sr$  было корректным.

5. Объявить символьную переменную, вывести на экран её значение, предварительно его изменив, обращаясь к переменной через указатель на указатель на `char`. Необходимо произвести объявление и инициализацию всех переменных, входящих в цепочку.

6. Вывести на экран значение вещественной константы, обращаясь к ней через указатель на указатель. Предварительно выполнить все необходимые объявления и инициализацию.

7. Объявить три целочисленные переменные. Проинициализировать две из них произвольными значениями. Объявить указатель на целое и настроить его на первую переменную. Объявить ссылку и связать её со второй переменной. В третью переменную сохранить сумму первых двух, обращаясь к ним через указатель и ссылку соответственно.

8. Объявить и проинициализировать три целочисленные переменные. Объявить три ссылки на целое и связать с этими переменными. Найти наибольшее из трёх значений переменных, обращаясь к ним через ссылки.

9. Объявить и проинициализировать две целочисленные переменные. Объявить ссылки на целое и связать с этими переменными. Объявить третью целочисленную переменную, проинициализировав её разностью квадратов первых двух. К переменным обращаться по ссылке.

10. Объявить и проинициализировать две целочисленные переменные. С одной связать ссылку с другой указатель. Удвоить значения переменных, обращаясь к ним через указатель и ссылку соответственно.

## **5.2. СТАТИЧЕСКИЕ МАССИВЫ**

Поименованную последовательность, однотипных элементов, расположенную в памяти подряд называют *массивом*.

Для объявления массива нужно указать тип данных его элементов, имя массива и количество элементов в квадратных скобках.

`<тип элементов массива> <имя массива>[<количество элементов>];`

Нумерация ячеек массива начинается с нуля, следовательно, номер последнего элемента массива на единицу меньше количества элементов в массиве.

Объявление массива целых чисел из пяти элементов:

```
int mas [5];
```

Количество элементов должно быть обязательно задано целочисленной *константой*. Так как в момент компиляции происходит выделение памяти под массив, следовательно, должно быть известно количество его элементов и это значение не должно изменяться, для обеспечения корректной работы.

В примере, приведённом выше была использована неименованная константа. Предпочтительнее предварительно объявить именованную константу, задав ей необходимое значение, как показано ниже.

```
int const N=5;  
int mas [N];
```

При необходимости внесения изменений в программу, связанных с размером массива, достаточно будет изменить значение константы при объявлении.

Задать значение элементов массива можно следующими тремя способами: при объявлении, в цикле и поэлементно. Рассмотрим каждый из них подробнее.

В первом случае, при инициализации массива при объявлении, после закрывающей квадратной скобки ставят знак = и перечисляют значения элементов массива через запятую. При этом весь список значений заключён в фигурные скобки. Например:

```
int mas [N]={1,2,3,4,5};
```

Если список значений содержит меньше элементов чем их количество, указанное в квадратных скобках, то указанными значениями будут проинициализированы

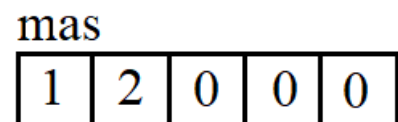


Рис. 25. Схематичное представление массива в памяти и с точки зрения языка C++, пример инициализации

ны первые элементы массива. Оставшиеся элементы получают значение по умолчанию для соответствующего типа данных. Например:

```
int mas [N]={1,2};
```

На рис. 25 схематично представлена такая инициализация массива. В случае, когда список инициализации содержит больше необходимого числа значений, на шаге компиляции возникает ошибка:

```
int mas1 [3]={1,2,3,4,5,6,7,8,9};// ошибка
```

Для инициализации массива в цикле и поэлементно необходимо научиться обращаться к элементам. Для этого существует два подхода.

В первом используют индексное выражение. Для этого после имени массива в квадратных скобках указывают индекс соответствующего элемента.

```
mas[3]=45;
```

Значение элемента с индексом три будет изменено на 45.

При инициализации элементов массива в цикле организуют перебор элементов, обычно от наименьшего индекса к наибольшему.

Объявим массив из пяти целых чисел и проинициализируем его в цикле значениями индексов соответствующих элементов.

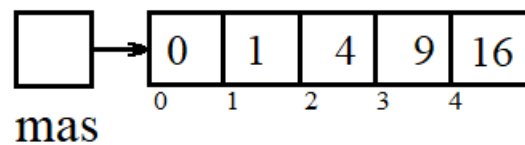
```
int mas[5];  
for (int i = 0; i < 5; i++)  
{mas[i] = i*i; cout << mas[i]<<" "};
```

Элементы массива будут проинициализированы значениями 0,1,2,3,4.

С точки зрения языка программирования C++ имя массива - это константный виртуальный указатель на его нулевой элемент. Как мы знаем, для обращения к адресуемому необходимо разыменовать указатель. \*mas = 8; Таким образом нулевой ячейке массива будет присвоено значение восемь. Для обращения к элементу массива с индексом *i* необходимо сначала получить его адрес. Для этого к адресу нулевого элемента прибавить соответствующее число. После чего разы-

меновать полученное значение.  $*(mas+i)=9$ ;  $i$ -ая ячейка массива получает значение девять. Таким образом можно обращаться к элементам массива в цикле, где переменная  $i$  будет принимать значения от нуля до номера последнего элемента. Перебор элементов будет корректен так как элементы в памяти расположены подряд. Важно понимать, что при этом значение указателя на нулевой элемент не изменится, происходит только вычисление адреса. Проинициализируем одномерный целочисленный массив квадратами индексов соответствующих элементов. Результат такой инициализации показан схематично на рис. 26.

```
int const N = 5;
int mas[5];
for (int i=0; i<N; i++)
{
    *(mas+i)=i*i;6
    printf("%i ",mas[i]);
}
```



*Рис. 26. Схематичное представление одномерного массива с точки зрения языка C++, после инициализации*

Рассмотрим подробнее расположение массива в памяти компьютера. Как было упомянуто выше, в языке программирования C++ имя массива интерпретируется как константный виртуальный указатель на его нулевой элемент. Константный - означает, что его значение неизменяемо, т.е. указатель нельзя перенастроить. Виртуальный, означает, что для его хранения место в память компьютера не выделяется, но с точки зрения языка программирования работа с именем массива ведётся по правилам работы с указателями, как было показано выше.

Ниже на рис. 27 схематично показано расположение массива в памяти компьютера и с точки зрения языка программирования C++. Организуя циклический перебор элементов массива важно внимательно отслеживать выход за его границы. При попытке обратиться к элементу с номером большим чем номер последнего элемента возникнет ошибка во время исполнения программы.



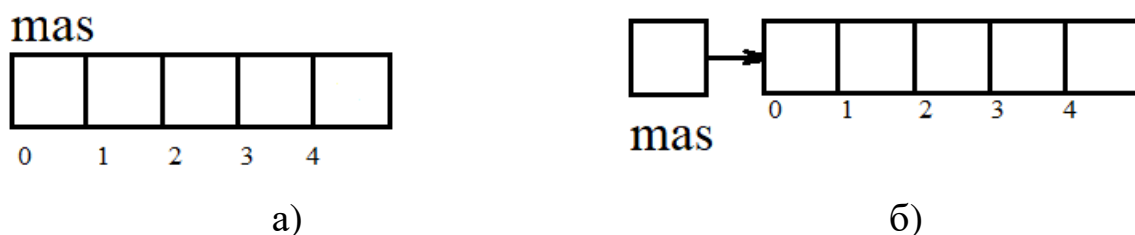


Рис. 27. Схематичное представление одномерного массива в памяти компьютера и с точки зрения языка C++: а - расположение одномерного массива в памяти компьютера; б - одномерный массив с точки зрения языка C++

### ***Вопросы для самоконтроля, подготовки к выполнению и защите лабораторной работы***

1. Что в программировании называют массивом?
2. Нумерация элементов массива в C++ начинается с нуля или единицы?
3. Что необходимо сделать для задания количества элементов одномерного статического массива?
4. Запишите синтаксис объявления одномерного массива без инициализации.
5. Назовите способы инициализации элементов массива.
6. Назовите и приведите примеры обращения к элементам массива.
7. Объявите одномерный целочисленный массив из десяти элементов, проинициализируйте его при объявлении.
8. Объявите одномерный целочисленный массив из десяти элементов, проинициализировав его элементы в цикле значениями их удвоенных индексов.
9. Объявите одномерный целочисленный массив из десяти элементов, проинициализируйте его в цикле вводом с клавиатуры.
10. Объявите одномерный целочисленный массив из десяти элементов, проинициализируйте его в цикле вводом с клавиатуры. Объявите указатель на целое. Настройте этот указатель на максимальный элемент массива.

### *Задания для самостоятельной работы*

1. Объявить и проинициализировать вводом с клавиатуры массив из 30 целочисленных элементов. Найти пять соседних элементов, сумма значений которых максимальна. Вывести на экран сумму и границы последовательности.

2. Объявить и проинициализировать вводом с клавиатуры массив из 20 целочисленных элементов. Уменьшить вдвое все элементы, оканчивающиеся цифрой четыре.

3. Объявить и проинициализировать любым способом массив из 20 целочисленных элементов. Найти число пар соседних элементов массива, являющихся четными числами.

4. Объявить и проинициализировать любым способом массив из 20 целочисленных элементов. Определить количество элементов, больших суммы всех элементов массива, вывести на экран найденные элементы и их номера.

5. Объявить и проинициализировать любым способом массив из 20 целочисленных элементов. Найти и вывести на экран элемент и его номер, наиболее близкий к среднему значению элементов массива.

6. По заданным вещественным числам  $a_0, a_1, \dots, a_{10}$  ( $a$  - массив). Вычислить значение многочлена  $a_{10}x_{10} + a_9x_9 + \dots + a_1x_1 + a_0$  и его производной в точке  $t$ .

7. Объявить и проинициализировать любым способом массив из 20 целочисленных элементов. Определить, имеются ли в массиве одинаковые элементы.

8. Дан массив, такой, что его значения образуют неубывающую последовательность. Несколько элементов, идущих подряд, могут быть равны между собой. Найти самую длинную серию повторяющихся элементов.

9. Даны два целочисленных массива. Найти наименьшее среди значений элементов первого массива, которые не входят во второй массив (по условию задачи хотя бы одно такое число есть).

10. Дан целочисленный массив, его элементы образуют неубывающую последовательность. Найти количество различных чисел в массиве.

11. Дан одномерный целочисленный массив. Определить и вывести на экран минимальный элемент массива и его номер, а также номер и значение элемента, являющегося минимальным без учета первого найденного элемента.

12. Дан одномерный целочисленный массив. Изменить знак максимального по модулю элемента массива.

13. Дан одномерный целочисленный массив Переменной  $t$  присвоить значение истина или ложь в зависимости от того, есть ли среди элементов массива число, являющееся степенью двойки, вывести на экран сообщение об этом в зависимости от значения переменной  $t$ .

14. Удалить из массива первый отрицательный элемент (если отрицательные элементы есть). Удаление из массива производить следующим образом. Под удалением элемента понимается исключение этого элемента из массива путем смещения всех следующих за ним элементов влево на 1 позицию и присваивание последнему элементу массива значения 0.

15. Дан одномерный целочисленный массив. Рассматривая его элементы как последовательность цифр десятичной записи некоторого неотрицательного целого числа, получить это число, сохранить в переменную и вывести на экран. (Например, массив {1,2,3} представляет число 123).

16. Дан одномерный целочисленный массив. Вставить в него заданное число перед последним четным элементом (если четные элементы есть). Под вставкой числа  $n$  в массив после  $k$ -го элемента понимается смещение всех элементов, начиная с  $(k+1)$ -го вправо на 1 позицию, и присваивание  $(k+1)$ -му элементу массива значения  $n$ . При этом значение последнего элемента массива теряется.

17. Дан одномерный целочисленный массив. Если в нём есть хотя бы одна пара соседних неотрицательных элементов, вывести на экран все элементы, следующие за первой такой парой.

18. Дан одномерный целочисленный массив. Вычислить и вывести на экран разность сумм его отрицательных и положительных

элементов. Дан одномерный целочисленный массив. Сдвинуть циклически на две позиции влево его элементы.

19. Вычислить  $y = x_1 + x_1x_2 + x_1x_2x_3 + \dots + x_1x_2\dots x_m$ , где  $x_i$  – элементы одномерного целочисленного массива,  $m$  – либо номер первого отрицательного элемента в этом массиве, либо номер последнего элемента, если в массиве не нашлось отрицательных элементов.

20. Найти сумму элементов массива, расположенных между максимальным и минимальным элементами (включительно).

### 5.3. МНОГОМЕРНЫЕ МАССИВЫ

Размерность массива определяется количеством квадратных скобок после его имени.

*<тип элементов имя массива> [<целочисленная константа1>][<целочисленная константа2>];*

Например, так объявлен двумерный массив:

```
int const M = 5;  
int const N = 3;  
int mas2[N][M];
```

Объявление трёхмерного массива.

```
int mas3[N][M][N];
```

Элементы многомерных массивов в памяти располагаются подряд построчно. При переходе к следующему элементу первым изменяется последний индекс.

`mas[0][0]=2;` - запись значения в ячейку с индексами 0, 0.

На рис. 28 схематично представлено расположение двумерного массива в памяти и с точки зрения синтаксиса языка C++.

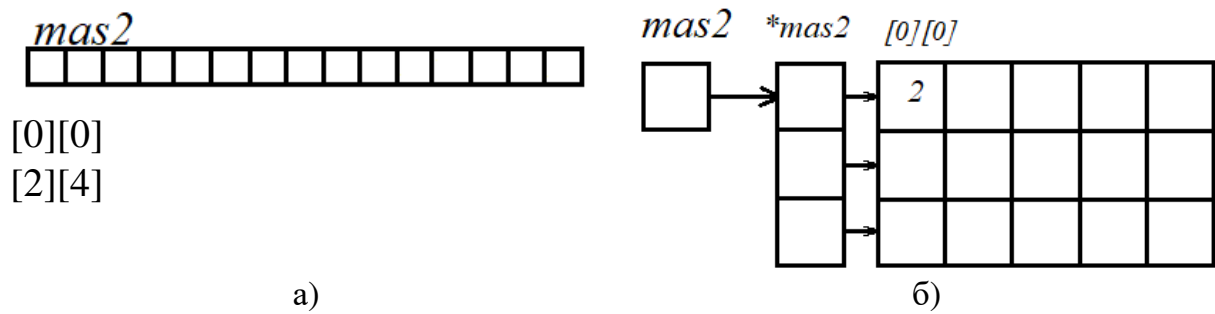


Рис. 28. Схематичное представление двумерного массива: а - расположение двумерного массива в памяти, б - представление массива с точки зрения языка C++

Для задания значений многомерного массива при объявлении их указывают в фигурных скобках списком через запятую.

```
int mas2[3][2] = {1,2,3,4,5,6 };
```

Приведём пример вывода элементов двумерного массива в цикле.

```
for (int i = 0; i < 3; i++)
{
for (int j = 0; j < 2; j++)
printf("%i ", mas2[i][j]);
printf("\n");
}
```

Изменим значения некоторых элементов используя имя массива как указатель.

```
mas2[1][1] = 2;
*(*(mas2+1)+2) = 3;
```

## 5.4. ДИНАМИЧЕСКОЕ РАСПРЕДЕЛЕНИЕ ПАМЯТИ

До этого момента работа велась с переменными, память под которые выделяется на этапе компиляции, такие переменные называют *статическими*.

Переменные память под хранение которых выделяется во время выполнения программы называют *динамическими*

*Динамически распределяемую память* называют кучей. Работа с динамическими переменными происходит через указатели.

Для выделения динамической памяти используют оператор `new`.

`new <тип данных> [<количество элементов>];`

или так:

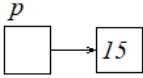
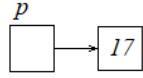

`new <тип данных> (<инициализатор>);`

В точку вызова оператор `new` возвращает указатель на начало области захваченной памяти.

В таблице 2 приведены примеры работы с данными память под которые захвачена с помощью оператора `new` и схематичное пояснение произведённых действий.

Таблица 2

### Работа с динамически захваченной памятью

Фрагмент кода	Схематичное представление
<code>int* p = new int(15);</code>	
Объявлен указатель на целочисленную переменную, выделена динамическая память под переменную типа <code>int</code> , в выделенную ячейку записано значение пятнадцать. Адрес начала захваченной области памяти записан в указатель <code>p</code> .	
<code>*p += 2;</code>	
Значение указуемого увеличено на 2	
<code>int* pM = new int[15];</code>	
Объявлен указатель на целочисленную переменную, выделена динамическая память под массив из 15 целых чисел. Адрес его записан в указатель <code>pM</code> .	

Фрагмент кода	Схематичное представление
<pre>for (int i = 0; i &lt; 15; j++) {pM[i] = i;}</pre>	<p>The diagram shows a pointer variable labeled 'pM' with a small square box next to it. An arrow points from this box to the first cell of a horizontal array of 15 cells. Each cell contains a number from 0 to 14, representing the indices of the array elements.</p>
Инициализация в цикле элементов динамического массива	

Динамическая память, захваченная при помощи `new` должна быть освобождена при помощи `delete`.

```
delete[] <указатель>;
```

```
delete p;
delete []pM;
```

### ***Вопросы для самоконтроля, подготовки к выполнению и защите лабораторной работы***

1. Что такое многомерный массив?
2. Какие данные называют статическими?
3. Какие переменные называют динамическими?
4. Каким образом осуществляется работа с динамическими переменными?
5. Запишите синтаксис объявления двумерного статического массива?
6. Каким образом можно проинициализировать двумерный статический массив?
7. Продемонстрируйте организацию перебора элементов двумерного целочисленного массива в цикле.
8. Приведите пример создания и инициализации двумерного динамического массива.
9. Приведите пример создания и инициализации трёхмерного динамического массива.
10. С помощью какого оператора освобождают динамическую память?

### *Задания для самостоятельной работы*

1. Поясните схематично следующие строки кода:

```
int i = 10;  
int *pi = &i;  
int *pi2 = int(10);  
int *pi3 = new int[10];
```

В следующих задачах необходимо объявить и проинициализировать двумерный динамический массив, размера  $n$  на  $m$ , где  $n$  – число строк,  $m$  – число столбцов.

1. Вычислить и вывести на экран сумму элементов двумерного массива.

2. Найти и вывести на экран максимальный элемент двумерного массива и его индексы.

3. Найти и вывести на экран минимальный элемент двумерного целочисленного массива и его индексы.

4. Заменить в двумерном целочисленном массиве все отрицательные элементы на минимальное значение массива.

5. Подсчитать и вывести на экран количество строк, не содержащих ни одного нулевого элемента.

6. Определить максимальное из чисел, встречающихся в массиве более одного раза.

7. В двумерном целочисленном массиве подсчитать количество столбцов, не содержащих ни одного отрицательного элемента.

8. Расположить строки двумерного массива в соответствии с ростом суммы элементов в этих строках.

9. Расположить строки двумерного массива в соответствии с ростом суммы положительных элементов в этих строках.

10. Расположить строки двумерного массива в соответствии с ростом суммы положительных четных элементов в этих строках.



## Лабораторная работа № 5

### УКАЗАТЕЛИ. ССЫЛКИ. МАССИВЫ. РАБОТА С ДИНАМИЧЕСКОЙ ПАМЯТЬЮ

#### *Цель работы*

Изучить принципы работы с указателями и ссылками в языке программирования C++. Получить навыки применения операций на указателях и ссылках, работы с одномерными и многомерными статическими и динамическими массивами, изучить способы создания, инициализации, перебора элементов в многомерных массивах, приёмов накопления.

#### *Постановка задачи*

1. Объявить и проинициализировать переменные базовых типов данных: `int`, `float`, `char`, `void`.

2. Объявить указатели и ссылки на все перечисленные выше тип данных, настроить их на объявленные переменные. Вывести значения переменных на экран, обращаясь к ним через указатели.

3. Привести примеры допустимых операций над указателями и ссылками. Пояснить смысл данных операций.

4. Объявить и проинициализировать следующие переменные и константы:

- константный указатель на `int`;
- указатель на целочисленную константу;
- указатель на указатель на `int`;
- константный указатель на указатель на `int`;
- указатель на константный указатель на целое;

Пояснить схематично организованные цепочки данных.

5. Объявить и проинициализировать одномерный целочисленный статический массив. Вычислить и вывести на экран сумму элементов этого массива, больших нуля.

6. Объявить двумерный статический массив вещественных чисел. Проинициализировать элементы массива любым способом.

7. Организовать вывод в виде таблицы элементов двумерного массива.

8. Создать двумерный целочисленный динамический массив. Задать значения элементов массива. В созданном массиве найти: мак-

симальное значение, минимальное значение, сумму всех элементов массива.

### ***Указания к выполнению работы***

1. Инициализацию переменных производить при объявлении. Объявить переменную типа `void`, убедиться в том, что такое объявление приводит к ошибке. Закомментировать строку объявления.

2. Объявленный указатель на `void`, настроить на целочисленную переменную, вывести на экран значение соответствующей переменной обращаясь к ней через указатель на `void`.

Все значения выводить в следующем порядке: значение переменной, значение переменной, обращаясь к ней через указатель, значение указателя.

3. Вопрос явного и не явного приведения типов был рассмотрен в главе базовые типы языка C++. При работе с указателями правила остаются теми же.

4. Необходимо использовать все операции, рассмотренные на лекции. В комментариях дать пояснение как выполняется операция.

5. Пояснить схематично организованные цепочки данных.

### ***Вопросы для самоконтроля, подготовки к выполнению и защите лабораторной работы***

1. Что в программировании называют указателем?
2. Для каких целей используют указатели?
3. Что такое ссылка?
4. Для чего используют ссылки?
5. Что в программировании называют массивом?
6. Приведите синтаксис и пример объявления статического массива?
7. Перечислите способы обращения к элементам массива?
8. Какие массивы называют динамическими, какие – статическими?
9. Объявите и проинициализируйте динамическую переменную.
10. Объявите и проинициализируйте динамический массив.

### *Задания для самостоятельной работы*

1. Объявить и проинициализировать вводом с клавиатуры одномерный массив вещественных чисел. Вычислить и вывести на экран разность квадратов максимального и минимального значения элементов этого массива.

2. Объявить одномерный целочисленный массив. Вычислить среднее арифметическое элементов, обращаясь к ним через имя массива, используя его как указатель.

3. Объявить два указателя на целое, настроить их на максимальный и минимальный элементы массива соответственно. Вывести на экран разность максимального и минимального элементов массива обращаясь к ним через указатели. Дополнительные переменные не использовать.

4. Объявить массив указателей на вещественный тип данных.

5. Настроить элементы массива указателей на минимальные элементы в каждой строке двумерного массива вещественных чисел.

6. Вывести на экран минимальные элементы строк массива, обращаясь к ним через настроенные на них элементы массива указателей.

7. Найти и вывести на экран индексы минимального элемента двумерного динамического массива.

8. Объявить указатель на указатель на целое, настроить его на последовательность из пяти указателей на целое, каждый из которых настроить на последовательность из 5 целых чисел.

9. В созданной структуре данных найти: максимальное значение среди сумм элементов в строках; минимальное значение среди произведений элементов в столбцах.

10. Вычислить квадрат разности максимального и минимального элементов, двумерного динамического массива.

## 6. СПОСОБЫ СОРТИРОВКИ МАССИВОВ

Массив называют отсортированным, если его значения расположены по возрастанию или убыванию.

Если входная последовательность упорядочена, говорят о лучшем случае входных данных.

Рассмотрим несколько алгоритмов сортировки данных.

### 6.1. СОРТИРОВКА ВЫБОРОМ

Суть алгоритма сводится к поиску (выбору) минимального элемента и обмена его местами с элементом, стоящим на первом месте.

Процесс повторяется для оставшейся части массива, найденный элемент будет обмену местами с элементом, стоящим на второй позиции и так далее, пока вся последовательность не будет отсортирована. Очевидно, что для реализации этого алгоритма необходимо использовать два цикла. В теле вложенного будет осуществлён поиск минимального элемента, в теле внешнего - обмен элементов местами и сдвиг индекса, отвечающего за номер элемента на место которого будет поставлен минимальный.

На рис. 29 представлена блок-схема алгоритма сортировки одномерного массива методом выбора.

Для осуществления обмена двух элементов местами необходимо задействовать дополнительную ячейку, того же типа, что и элементы массива. Скопировать в эту ячейку первое значение, минимальное

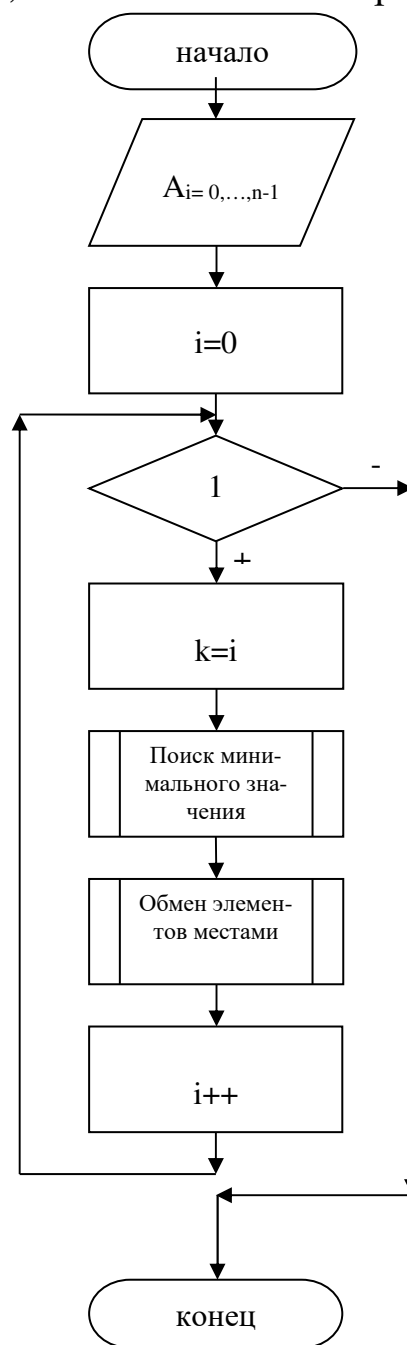


Рис. 29. Блок-схема алгоритма сортировки массива методом выбора

значение записать на место первого элемента, значение из дополнительной переменной записать в ячейку на место минимального значения.

```
int tmp = mas[0];
mas[0] = mas[min];
mas[min] = tmp;
```

На рис. 30 показан схематично обмен элементов местами.

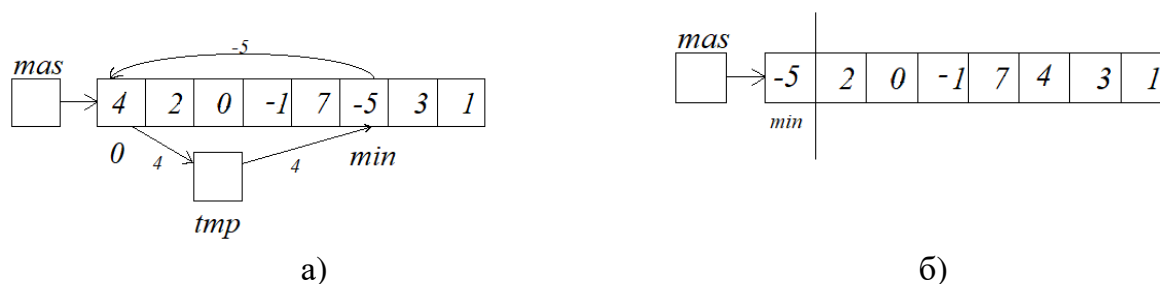


Рис. 30. Обмен элементов местами: а – обмен элементов местами, б – результат обмена

На рис. 31 показан результат итераций внешнего цикла, при сортировке массива методом выбора.



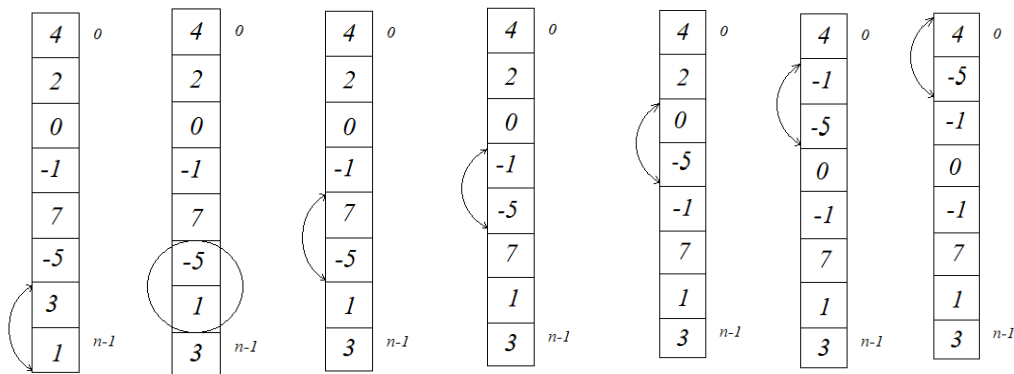
Рис. 31. Результат итераций внешнего цикла

## 6.2. СОРТИРОВКА ПУЗЫРЬКОМ

Название метода следует из аналогии со всплывающим пузырьком. Так же самый лёгкий (наименьший) элемент всплывает наверх – на позицию с наименьшим индексом.

Для понимания алгоритма массив удобнее представлять расположенным сверху вниз.

Алгоритм реализуется с помощью двух циклов. В теле вложенного осуществляется перебор элементов от максимального индекса в сторону уменьшения индексов. При этом происходит сравнение двух стоящих рядом элементов. И обмен элементов местами, если элемент с меньшим индексом, больше следующего элемента (при сортировке по возрастанию). На рис.32 схематично показан такой перебор элементов.



*Рис. 32. Первая итерация внешнего цикла*

После такого перебора элементов на нулевой позиции стоит минимальный элемент. Таким образом, следующий проход осуществляется не до нулевого элемента, а до первого. И так далее.

Легко заметить, что переменная внешнего цикла изменяется от 0 до  $n-1$ , где  $n$  – количество элементов в массиве, а переменная вложенного цикла от  $n-1$  до значения переменной внешнего цикла.

Блок-схема алгоритма сортировки массива пузырьком показана рис. 33.

Внеся в алгоритм несколько изменений можно увеличить скорость сортировки.

1. Использование флага сортировки. Если за одну итерацию внешнего цикла не произошло ни одной перестановки, значит массив отсортирован и в дальнейшем переборе элементов нет смысла. Для того чтобы осуществить проверку на наличие перестановок достаточно ввести логическую переменную.

Задать значение этой переменной *true* до входа в цикл, изменить значение на *false* при перестановке элементов местами. Проверить значение логической переменной после выхода из вложенного цикла. Если значение не изменилось, значит перестановок не было и массив отсортирован. В этом случае осуществляют досрочный выход из цикла.

1. В случае если перестановки были и массив не отсортирован, имеет смысл запомнить индекс последней перестановки. Элементы с индексами меньше запомненного уже отсортированы и следующую итерацию нет смысла продолжать дальше индекса последней перестановки.

2. Если после прохода снизу-вверх осуществить проход сверху-вниз, изменив знак в операции сравнения, то скорость сортировки также увеличится, т.к. за один такой проход наибольший элемент встанет на свою позицию. Такое чередование направлений прохода называют, Шейкер-сортировкой.

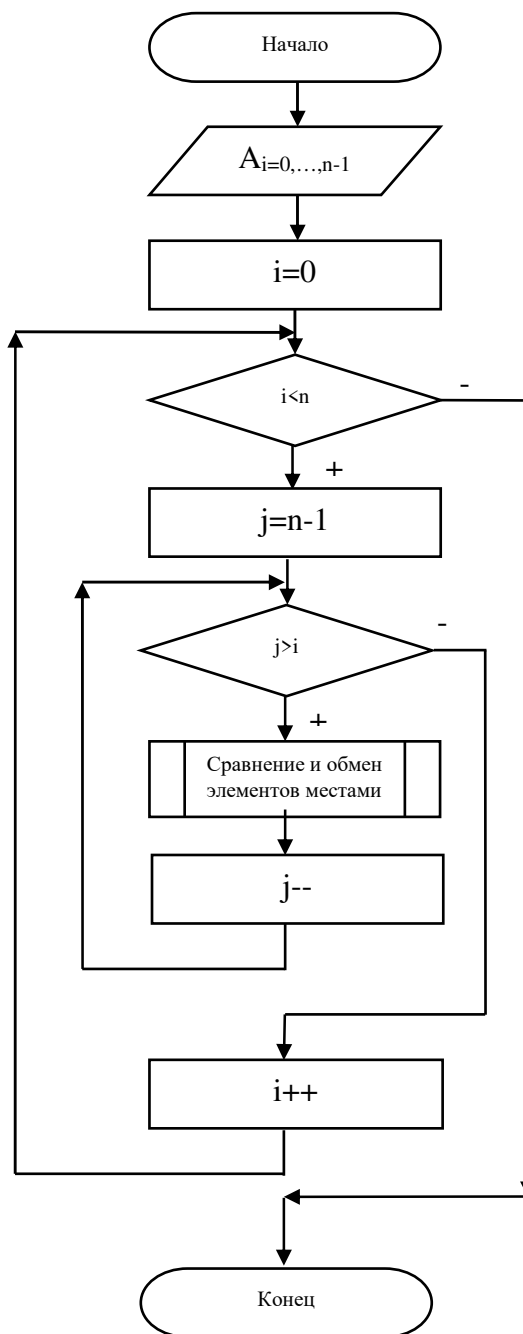


Рис. 33. Блок-схема алгоритма сортировки пузырьком

### 6.3. СОРТИРОВКА ПРОСТОЙ ВСТАВКОЙ

Сортировка методом простой вставки предполагает поиск позиции текущего элемента в отсортированной части. Рассмотрим  $i$ -ый шаг алгоритма. Все элементы левее  $i$ -го (с меньшими индексами) отсортированы. Т.е., если мы говорим о сортировке по возрастанию, каждый следующий не меньше предыдущего (больше либо равен предыдущему). Если мы говорим о сортировке по убыванию, то каждый следующий элемент должен быть меньше либо равен предыдущему. Задача  $i$ -го шага - найти место  $i$ -му элементу в отсортированной части. Для этого производят поочерёдные сравнения  $i$ -го элемента с предыдущими и сдвиг сравниваемого элемента вправо при необходимости. Процесс завершается, в двух случаях: если перебрана вся последовательность и, если предыдущий элемент оказался меньше  $i$ -го.

Таким образом,  $i$ -ый элемент находит своё место – вставка завершена. Далее во внешнем цикле осуществляется переход к следующему элементу, т.е. увеличение параметра  $i$ . На первом шаге алгоритма считаем отсортированную часть из одного элемента с наименьшим индексом. Таким образом, внешний цикл начинается с единицы, вложенный - с номера элемента на единицу меньше.

На рис. 34 представлена блок-схема алгоритма сортировки массива методом вставки

Во внешнем цикле осуществляет перебор элементов от второго до последнего, во вложенном цикле осуществляется вставка текущего элемента в отсортированную часть. Соответственно, параметр внешнего цикла изменяется от 1 до  $n-1$ , Параметр вложенного цикла изме-

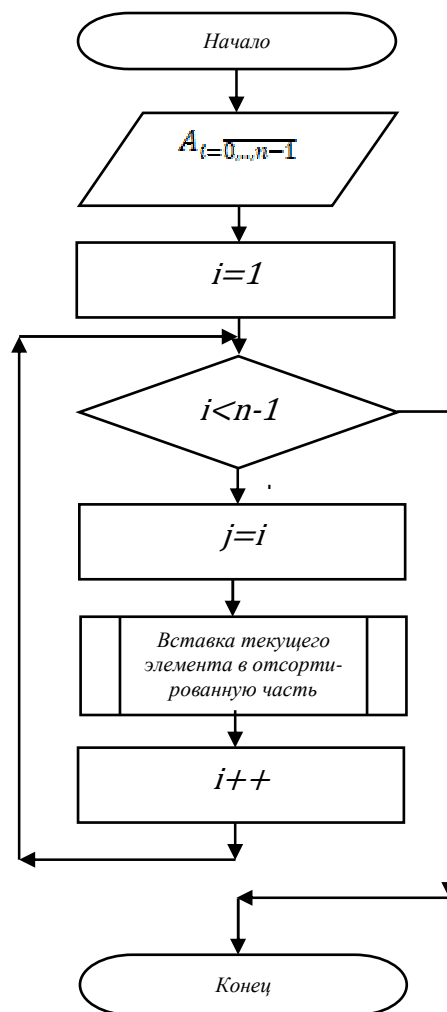


Рис. 34. Блок-схема алгоритма сортировки простой вставкой



няется от значения на единицу меньшего значения параметра внешнего цикла до нуля либо до того, как будет найдено место  $i$ -го элемента.

На рис. 35 схематично показан результат итераций внешнего цикла, т.е. вставка  $i$ -го элемента в отсортированную часть. Вертикальная черта означает границы отсортированной и не отсортированной частей массива.

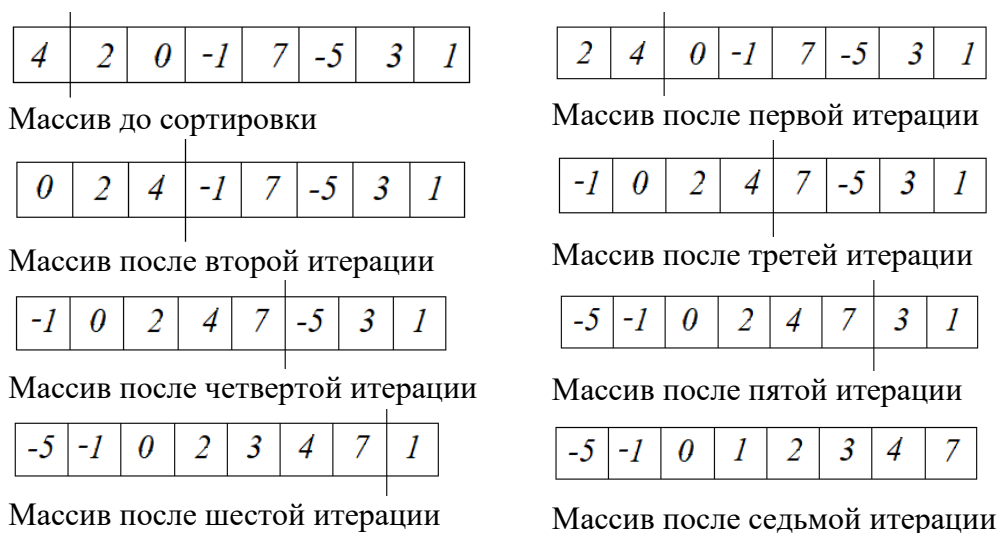


Рис. 35. Схематичное представление результата итераций внешнего цикла алгоритма сортировки простыми вставками.

## Лабораторная работа № 6

### АЛГОРИТМЫ СОРТИРОВКИ ДАННЫХ

#### ***Цель работы***

Закрепление навыков реализации циклических алгоритмов, написания программного кода по графическому представлению алгоритма. Изучение алгоритмов сортировки данных.

#### ***Постановка задачи***

Реализовать графическое представление (в виде блок-схемы) и программный код алгоритмов сортировки массивов данных: сортировка выбором, пузырьком, простыми вставками.

## ***Вопросы для самоконтроля, подготовки к выполнению и защите лабораторной работы***

1. В каком случае массив называют упорядоченным?
2. Перечислите изученные алгоритмы сортировки.
3. Перечислите изменения алгоритма сортировки пузырьком, позволяющие увеличить скорость сортировки.
4. В чём заключается Шейкер сортировка?
5. Что нужно изменить в алгоритме сортировки выбором, чтобы была реализована сортировка по убыванию?
6. Каково назначение флага в алгоритме сортировки пузырьком?
7. Какие дальнейшие действия нужно произвести с запомненным индексом последнего обмена для увеличения скорости сортировки в алгоритме сортировки пузырьком?
8. Какое начальное значение у переменной вложенного цикла в алгоритме сортировки вставками?
9. В каком случае вставка  $i$ -го элемента считается завершённой в алгоритме сортировки вставками?
10. В каком случае говорят о наилучшем случае входных данных для алгоритма сортировки?

## ***Задания для самостоятельной работы***

- Реализовать на языке программирования C++:
- 1) Алгоритм сортировки пузырьком с использованием флага.
  - 2) Алгоритм сортировки пузырьком с использованием запоминания индекса последнего обмена.
  - 3) Шейкер сортировку.
  - 4) Реализуйте алгоритм сортировки пузырьком с использованием трёх улучшений, приведённых в теоретической части раздела.
  - 5) Алгоритм сортировки Шелла.
  - 6) Алгоритм гномьей сортировки.
  - 7) Алгоритм пирамидальной сортировки.
  - 8) Сортировку слиянием на двух упорядоченных целочисленных массивах.
  - 9) Пирамидальную сортировку
  - 10) Сортировку подсчётом.

## 7. ПОЛЬЗОВАТЕЛЬСКИЕ ТИПЫ ДАННЫХ. СТРУКТУРЫ

Пользовательскими называют типы данных, которые описывает программист по определённым правилам, исходя из условий конкретной задачи.

К пользовательским типам относят структуры.

**Структура** – пользовательский тип данных, объединяющий под одним именем разнотипные переменные, называемые полями.

Описание типа – структура начинается с ключевого слова `struct` и указания имени соответствующего типа, после чего в фигурных скобках следует описание полей. Описание завершается точкой с запятой.

```
struct имя типа {описание полей};
```

```
struct ST_Example  
{  
    int pole1;  
    float pole2;  
    char pole3[10];  
};
```

После описания типа данных возможно объявление переменной этого типа. Объявление переменной типа структура подчиняется тем же правилам, что и объявление переменных базового типа.

```
<имя типа> <имя переменной>;
```

Объявим переменную описанного выше типа:

```
ST_Example Ex1, Ex2;
```

Объявление переменных может быть совмещено с описанием типа данных. В этом случае, после закрывающей фигурной скобки, до точки с запятой перечисляют необходимые переменные описанного типа.

```

struct ST_Example
{
    int pole1;
    float pole2;
    char pole3[10];
} Ex;

```

Таким образом после описания типа `ST_Example` объявлена переменная `Ex`.

Инициализация полей объявленных переменных возможна двумя способами: списком и через уточняющие идентификаторы.

Разберём подробнее эти способы.

В первом случае нужно в фигурных скобках перечислить через запятую необходимые значения полей в порядке их объявления.

```

struct ST_Example
{
    int pole1;
    float pole2;
    char pole3[10];
} Ex{1, 2.3, "name"};
ST_Example Ex1{ 1, 3.14, "name1" },
            Ex2 = {10, 2.4, "name1"};

```

Во втором подходе обращаются к каждому полю, следующим образом: после имени переменной типа структура ставят точку и указывают имя поля.

*<имя переменной>.<имя поля>*

```
Ex1.pole1 = 9;
```

При необходимости сохранить значения полей одной структуры в поля другой структуры того же типа используют операцию присваивания, либо копируют каждое поле в отдельности.

```
Ex1 = Ex2;  
Ex1.pole1 = Ex2.pole1;  
Ex1.pole2 = Ex2.pole2;
```

Ниже приведены примеры объявления массива структур, и указателя на структуру, указатель проинициализирован адресом переменной, объявленной ранее.

```
ST_Example massiv[5];  
ST_Example* pST = &Ex1;
```

## **Лабораторная работа № 7**

### **ПОЛЬЗОВАТЕЛЬСКИЕ ТИПЫ ДАННЫХ. СТРУКТУРЫ**

#### ***Цель работы***

Познакомиться с пользовательским типом данных – структура. Изучить объявление, инициализацию, обращение к полям, научиться работать с массивами структур.

#### ***Постановка задачи***

- описать тип данных STUDENT, представляющий собой структуру со следующими полями: фамилия и инициалы, номер группы, успеваемость, поле успеваемость задать массивом из 10 целых чисел;
- объявить массив из десяти элементов типа STUDENT;
- проинициализировать массив вводом с клавиатуры;
- выполнить сортировку массива структур в алфавитном порядке;
- вывести на экран фамилии и номера групп студентов, чей средний балл выше 4.0, если таких студентов нет, вывести соответствующее сообщение;
- объявить указатель тип STUDENT, настроить его на элемент массива с наивысшим средним баллом.

## **Вопросы для самоконтроля и подготовки к защите лабораторной работы**

1. Тип данных в языке программирования C++ является базовым или пользовательскими?
2. Напишите синтаксис и приведите пример описания структуры, объявления переменных описанного типа.
3. Каким образом можно проинициализировать объявленную структуру?
4. Каким образом можно задать значение полей структуры при объявлении?
5. Каким образом можно обратиться к полям структуры?
6. Возможно ли изменение значений проинициализированных полей структуры?
7. Объявите и проинициализируйте массив структур?
8. Возможно ли объявление указателя в качестве поля структуры, указателя на описываемый тип?
9. Продемонстрируйте объявление указателя на структуру?
10. Чем тип данных структура отличается от типа данных массив, что между ними общего?

## **Задания для самостоятельной работы**

1. Описать структуру *Point*. Написать программу, которая получает на вход количество точек -  $n$ , и последовательность их координат. Вывести на экран координаты точки, наиболее удалённой от начала координат.
2. Описать структуру *Zavod* содержащую следующую информацию фамилия, средний возраст, специальность, средний оклад. Объявить и проинициализировать массив из  $N$  таких структур. Вывести на экран информацию о количестве слесарей и токарей.
3. Багаж пассажира характеризуется количеством вещей и общим весом. Описать соответствующую структуру и создать массив из  $N$  таких структур. Проинициализировать массив. Вывести на экран информацию о пассажирах с самым большим багажом по числу и по весу.
4. Описать структуру *Zavod* содержащую следующую информацию фамилия, средний возраст, специальность, средний оклад.

Объявить и проинициализировать массив из  $N$  таких структур. Вывести на экран номера заводов, где средний возраст выше 35 лет.

5. Описать структуру *Zavod* содержащую следующую информацию фамилия, средний возраст, специальность, средний оклад. Объявить и проинициализировать массив из  $N$  таких структур. Найти количество заводов, где средний оклад по заводу выше среднего по всем заводам. Вывести на экран это значение и всю информацию по этим заводам.

6. Багаж пассажира характеризуется количеством вещей и общим весом. Описать соответствующую структуру и создать массив из  $N$  таких структур. Проинициализировать массив. Определить есть ли среди пассажиров такой, у которого самый большой багаж по количеству и наименьший по весу.

7. Багаж пассажира характеризуется количеством вещей и общим весом. Описать соответствующую структуру, создать массив из  $N$  таких структур. Проинициализировать массив. Определить есть ли среди пассажиров такой, у которого самый маленький багаж по числу и по весу.

8. Имеется информация по итогам экзаменов в институте всего в списке  $N$  человек. По каждому из студентов имеются следующие сведения: фамилия, оценка по математике, оценка по информатике и оценка по физике. Описать соответствующую структуру, объявить и проинициализировать массив из  $N$  элементов. Вывести на экран количество и фамилии отличников.

9. Имеется информация по итогам экзаменов в институте всего в списке  $N$  человек. По каждому из студентов имеются такие сведения: фамилия, оценка по математике, оценка по информатике и оценка по физике. Описать соответствующую структуру, объявить и проинициализировать массив из  $N$  элементов. Вывести на экран количество и фамилии отличников.

10. Имеется информация по итогам экзаменов в институте всего в списке  $N$  человек. По каждому из студентов имеются такие сведения: фамилия, оценка по математике, оценка по информатике и оценка по физике. Описать соответствующую структуру, объявить и проинициализировать массив из  $N$  элементов. Вывести на экран количество и фамилии круглых двоечников.

## 8. ФУНКЦИИ

В некоторых случаях часть кода удобнее оформить в виде функции. Функция решает отдельную небольшую задачу. После правильного написания кода функции, к ней можно обратиться в любом месте программы. Таким образом, если планируется неоднократный вызов функции, нет необходимости дублировать код, достаточно обратиться к уже написанному.

Итак, функцией называют именованную последовательность описаний и операторов, направленных на решение некой задачи. При использовании функций выделяют три этапа: объявление, определение и вызов.

До этого момента мы работали только с `main()` функцией. Любая программа на C++ должна содержать `main()` функцию. Это точка входа в программу. Т.е. начало выполнения программы.

Функции, созданные пользователем, должны быть предварительно объявлены и определены.

Объявление функции (или заголовок функции) содержит имя функции, тип возвращаемого значения, список формальных параметров в круглых скобках. Причём список параметров может отсутствовать, а наличие скобок обязательно. Как и любое другое объявление в C++, объявление функции заканчивается точкой с запятой.

*[<класс памяти>] тип возвращаемого значения <имя функции> ([список формальных параметров]);*

Функция обязательно должна быть объявлена до её вызова.

Определение функции, т.е. написание последовательности объявлений и действий, называемых телом функции, может быть совмещено с объявлением функции или вынесено в отдельный модуль. В любом случае, объявление функции должно содержаться в модуле, в котором она будет вызвана.

Определение функции содержит заголовок функции и тело функции – последовательность действий, заключённых в фигурные скобки.

Разберём подробнее объявление функции.



Класс памяти задаёт область видимости функции. Если при объявлении класс памяти не указан, будет использован класс памяти по умолчанию для функций - `extern`.

Функции, объявленные классом памяти `extern`, доступны для вызова во всех модулях программы.

Функции, объявленные с классом памяти `static`, доступны для вызова только том модуле, в котором они определены.

Тип возвращаемого значения - это тип значения, которое будет возвращено из функции в точку её вызова. Если из функции не планируется возвращать значение, то в качестве типа возвращаемого значения указывают тип `void`. В любом случае, тип возвращаемого значения должен быть указан.

Имя функции может быть любым, отвечающим требованиям именования в C++. После имени следуют обязательные круглые скобки. Они могут быть пустыми или содержать список формальных параметров.

Параметры перечисляются через запятую. При объявлении в круглых скобках можно указать только тип параметров, при определении нужно указывать тип и имя параметров.

Рассмотрим несколько примеров.

```
void f1();
```

Имя объявленной функции - `f1`. Тип возвращаемого значения – `void`, это значит, что функция не будет возвращать значения. Список формальных параметров пуст, т.е. функция не будет принимать данные для обработки.

Другой пример возможного объявления:

```
float f2 (float, float);
```

Здесь объявлена функция с именем `f2`, принимающая два вещественных параметра типа `float` и возвращающая в точку вызова значение типа `float`. В объявлении имена формальных параметров не указаны, при описании такой функции имена параметров нужно указать. Для возвращения значения из функции необходимо использовать оператор `return`. После оператора `return` должно быть указано

возвращаемое значение, если функция имеет тип возвращаемого значения отличный от `void`, как в предыдущем примере. Определение функции `f2` может быть таким:

```
float f2 (float c, float d){ return c+d;};
```

Определение функции `f1` таким:

```
void f1 () {cout << "это функция f1";}
```

Для вызова функции нужно указать имя функции и в круглых скобках список фактических параметров. Тип фактических параметров должен соответствовать типу формальных параметров.

Вызов функций `f1` и `f2`:

```
f1 ();  
f2 (1.2,3.4);
```

В точку вызова функции `f2` будет возвращено значение, в данном случае сумма параметров. Чтобы иметь возможность дальнейшего использования этого значения, его необходимо сохранить в переменной. Например:

```
int S = f2 (1.2,3.4);
```

Если возвращаемое значение не планируется использовать в дальнейших вычислениях и достаточно его вывода на экран, возможен такой подход:

```
cout << f2 (1.2,3.4);
```

Передать параметры в функцию можно тремя способами:

1. по значению;
2. по указателю;
3. по ссылке.

При передаче параметров в функцию по значению, в области памяти функции создаётся локальная копия фактических параметров. Изменение этих значений внутри функции не влечёт за собой изменения

внешних значений. Изменения затрагивают только локальную копию, недоступную после выхода из функции. Например:

```
int f1(int a)
{
    a++;
    return ++a;
}
```

```
int main()
{
    int a = 3;
    int b = f1(a);
}
```

В переменную `b` будет сохранено возвращаемое значение функции `f1`, переменная `b` примет значение равное пяти, значение переменной `a`, останется неизменным.

При передаче значений по указателю и ссылке в область памяти функции будут переданы адреса соответствующих переменных. Обращение к ним в теле функции приведёт к переходу по адресу. Соответственно, изменение значений в теле функции приведёт к изменению значений, переданных в функцию. Например:

```
int f2(int *b)
{
    (*b)++;
    return ++(*b);
}
```

```
int main()
{
    int b = 3;
    int* pb = &b;
    int b1 = f2(pb);
}
```

В приведённом примере значение переданной в функцию переменной будет увеличено на два. Таким образом значение переменных `b` и `b1` будут равны пяти.

При передаче параметра по ссылке нет необходимости размысливать переданное значение, что уменьшает вероятность возникновения ошибок.

```

int f3(int& c)
{
    (++c)++;
    return ++c;
}

int main()
{ int c = 3;
  int c1 = f3(c);
}

int f4(int a, int* b, int& c)
{a++; (*b)++; c++;
return a + *b + c;
}

int main()
{int a = 3;
int b = 3;
int* pb = &b;
int c = 3;
int S = f4(a,pb,c);
}

```

Разберём пример подробнее. В теле функции значения всех, переданных параметров увеличиваются на единицу. Таким образом, возвращаемое значение равно 12. При этом значение внешних переменных *b* и *c* увеличатся на единицу, значение переменной *a* останется неизменным.

Для передачи массива в функцию необходимо передать два параметра: адрес его начала и количество элементов. Так как передача массива в функцию всегда осуществляется по указателю, любые действия, произведённые с элементами массива в теле функции, осуществляются с элементами внешнего массива и возвращать массив из функции, после внесённых изменений, например, после сортировки, не нужно. При работе с многомерными массивами необходимо передавать в функцию информацию о всех размерностях. Внутри функции многомерный массив рассматривается как одномерный. Пример передачи массива в функцию:

```

void Q_S(int* mas, int first, int last)// объявление
    {. . .};// тело функции

```

```

. . .
int const n = 10;
int mas[n] = {1,-2,3,-4,5,-1,2,-3,4,-5};
Q_S(mas, 0, n-1); // вызов функции

```

### ***Параметры со значениями по умолчанию***

Параметры по умолчанию получают своё значение уже при объявлении функции. При вызове функции эти параметры можно не указывать. В этом случае будет использовано значение по умолчанию.

Параметры со значением по умолчанию обязательно должны находиться в конце списка формальных параметров. В случае, когда таких параметров несколько и один из них опущен при вызове, должны быть опущены и все следующие за ним параметры, следовательно, они примут значение по умолчанию. Например.

```

void f (int a, int b=1, int c=0 ); // объявление ф-ции
. . .
f(1); // первый вызов функции f.
f(1,2); // второй вызов функции f.

```

При первом вызове функции *f* параметры *b* и *c* принимают значение по умолчанию. При втором вызове функции *f* параметр *b* принимает значение «2», параметр *c* принимает значение по умолчанию.

## **Лабораторная работа № 8**

### **ФУНКЦИИ**

#### ***Цель работы***

Изучение принципов работы с функциями в языке программирования C++. Получение навыков объявления, определения, вызова функций, изучение способов передачи параметров в функции, работы с возвращаемым из функции значением.

#### ***Постановка задачи***

Написать и привести несколько различных примеров вызова следующих функций:

1. сложения двух целочисленных переменных;
2. поиска минимального значения;
3. сложения элементов целочисленного одномерного массива;
4. сортировки одномерного числового массива;
5. вывода элементов одномерного целочисленного массива на экран.

### ***Указания к выполнению работы***

1. Функция получает два целочисленных параметра, возвращает их сумму.
2. Функция получает три целочисленных параметра, возвращает наименьшее из них.
3. Функция получает на вход адрес первого элемента массива и количество элементов в массиве, возвращает целое число - сумму элементов в массиве.
4. Функция получает на вход данные о массиве, в качестве возвращаемого значения указать `void`.
5. Функция получает на вход данные о массиве, в качестве возвращаемого значения указать `void`.

### ***Вопросы для самоконтроля и подготовки к защите лабораторной работы***

1. Что такое функция?
2. Как правильно объявить функцию?
3. Что такое «класс памяти»?
4. Что называют возвращаемым из функции значением?
5. Что называют телом функции?
6. Какие параметры называют формальными?
7. Возможно ли объявить и описать функцию без параметров?
8. Какие параметры называют фактическими?
9. Перечислите способы передачи параметров в функцию? Поясните каждый из этих способов
10. Что нужно сделать для вызова функции?

## ***Задания для самостоятельной работы***

Опишите следующие функции и приведите несколько примеров её вызова (с различными значениями входных параметров).

1. Инициализации одномерного целочисленного массива.
2. Вывода на экран значений двумерного массива в табличном виде.
3. Вывода на экран столбца таблицы умножение на заданное в качестве параметра число.
4. Вычисление суммы минимальных значений в столбцах двумерного массива.
5. Вычисление суммы максимальных значений в строках двумерного массива.
6. Отсортировки методом выбора одномерного целочисленного массива.
7. Отсортировки методом простой вставки двумерного целочисленного массива.
8. Вычисления *факториала* заданного числа.
9. Вычисления суммы первых  $n$  натуральных чисел, где  $n$  – параметр.
10. Вычисления разности квадратов максимального и минимального значений в одномерном целочисленном массиве.

## **9. ЛИНЕЙНЫЕ ДИНАМИЧЕСКИЕ СТРУКТУРЫ ДАННЫХ**

Под структурой данных понимают множество данных и множество связей между ними.

Динамические структуры данных - это структуры, в которых количество элементов и связи между этими элементами могут изменяться во время выполнения программы. Такие структуры можно разделить на две группы по типу организации связи между элементами. Структуры могут быть линейные и нелинейные. Рассмотрим линейные структуры данных. Примером таких структур служат списки, очереди, стеки, деки и т.д.

*Списком называют* динамическую структуру данных, элементы которой связаны между собой определённым образом и не обязатель-

но расположены в памяти подряд. Размер структуры и порядок следования элементов, могут изменяться во время выполнения программы.

Динамические структуры данных используют, в ситуациях, когда заранее неизвестен объём данных, с которыми предстоит работать и предполагается изменение (увеличение или уменьшение) их количества во время исполнения программы. Связь элементов динамической структуры реализуют через указатели.

Линейные списки могут быть однонаправленными и двунаправленными, и те, и другие могут быть кольцевыми (закольцованными, замкнутыми).

Список называют кольцевым, если его последний элемент указывает на первый.

Если кольцевой (или закольцованный) список - двунаправленный, то каждый элемент указывает на следующий и предыдущий, соответственно, последний элемент настроен на первый и предпоследний, а первый элемент указывает на второй и на последний.

Для организации такой структуры данных каждый элемент списка должен содержать два вида полей: информационные поля указатели, служащие для связи между элементами. Количество информационных полей и их типы определяют исходя из условий конкретной решаемой задачи. Поля указатели должны иметь тип данных - указатель на элемент списка. Количество полей указателей определяется видом списка, один указатель для однонаправленных и два для двунаправленных.

Так как элемент списка объединяет в себе разнотипные поля в языке программирования C++ его описывают типом данных - структура. В данном случае, под словом структура понимают тип данных. В употребляемых выше словосочетаниях – «структура данных», имеется ввиду подход к организации данных.

Рассмотрим пример описания элемента динамического списка. Опишем структуру с двумя полями: информационным и полем указатель на следующий элемент. Информационное поле будет целочисленным.

```
struct elem
{int inf;
  elem * p;};
```



Поле `p` имеет тип – указатель на элемент (где элемент – это описываемая структура). Схематично элемент списка показан на рис.36, представленном ниже.

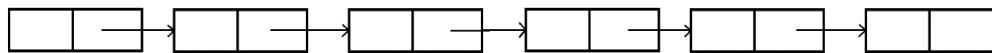


*Рис. 36. Схематичное отображение элемента динамического списка*

Создадим и проинициализируем элемент описанного выше типа.

```
elem E1 = { 2, NULL };
```

На рис. 37 схематично показана структура данных - линейный список, который можно организовать, используя в качестве элементов описанный тип `elem`.



*Рис. 37. Схематичное представление линейного списка*

Из схемы, представленной на рис. 37 понятно, что для обращения к элементам такой структуры необходим как минимум один указатель на её начало для обращения к элементам.

Элемент списка не может быть именованным, как в предыдущем примере (`elem E1 = { 2, NULL };`), Поэтому работа по созданию динамического списка начинается с объявления указателя на список. Этот указатель должен иметь тип указатель на эту структуру. Для дальнейшей корректной работы может понадобиться указатель на текущий элемент. Результат объявления настройки таких указателей показан на рис.38.

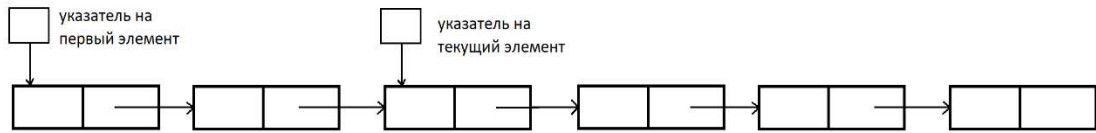


Рис. 38. Схематичное представление линейного списка и указателей, настроенных на первый и текущий элементы

Если список кольцевой, то последний элемент необходимо настроить на первый (рис. 39).

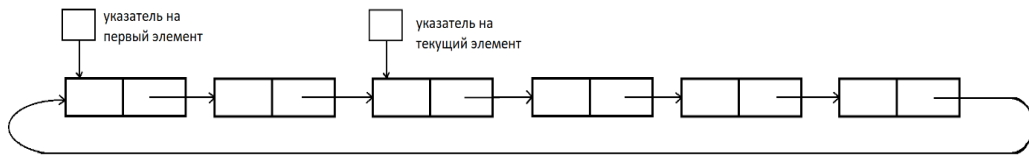


Рис. 39. Однонаправленный кольцевой список

Если список двунаправленный, каждый его элемент должен содержать два поля указателя, настроенные на следующий и на предыдущий элементы.

```
struct elem
{
    int data;
    elem* prev;
    elem* next;
};
```

Схематично элемент типа `struct elem` представлен на рис. 40.

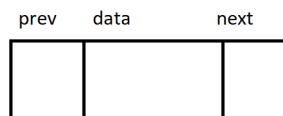


Рис. 40. Элемент двунаправленного списка

Если двунаправленный список – кольцевой, предыдущим элементом для первого является последний, а следующим для последнего – первый. Это наглядно показано на рис.41.

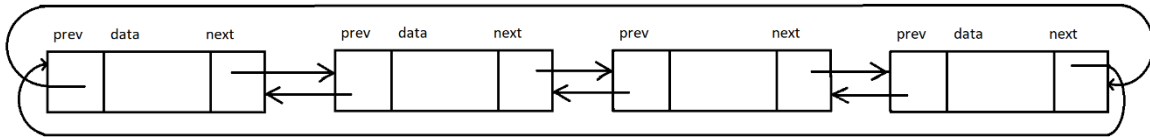


Рис. 41. Двухнаправленный кольцевой список

Для работы с разобранными выше динамическими структурами данных необходимо определить следующие операции:

- создание первого элемента;
- добавление элемента в начало или конец списка;
- поиск элемента по ключу;
- вставка элемента по ключу;
- упорядочивание по ключу;
- удаление элемента из начала или конца списка;
- удаление элемента по ключу.

При добавлении и исключении элементов из списка, необходимо осуществлять проверку списка на пустоту следующим образом:

`head != NULL`

где `head` – указатель на первый элемент списка (иначе называемый головой или вершиной списка).

Ясно, что алгоритм добавления элемента в список будет зависеть от того первый это элемент списка или нет. То же касается и удаления. Предварительно нужно убедиться есть ли элементы в списке.

### *Добавление элемента в однонаправленный линейный список*

*Добавление первого элемента:*

- захватить память под элемент;
- заполнить соответствующим образом информационные поля, поле указатель настроить на NULL (рис. 42).

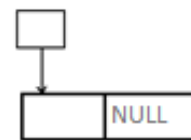
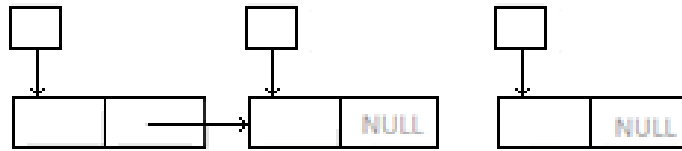


Рис. 42. Создан первый элемент списка

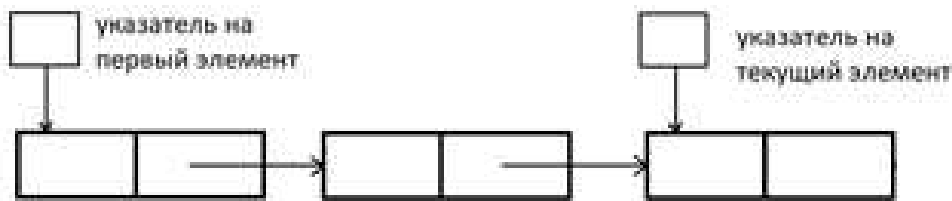
*Добавление последующих элементов:*

- захватить память под элемент (рис. 43);



*Рис. 43. Захвачена память под новый элемент, который будет добавлен к существующему списку*

– настроить соответствующим образом поля указатели данного и предыдущего элемента списка, настройка полей указателей будет зависеть от того куда будет добавлен новый элемент, в голову или хвост списка (рис.44);



*Рис. 44. Элемент добавлен в хвост линейного динамического списка*

### ***Вставка элемента по ключу***

Перед тем как осуществлять вставку необходимо реализовать поиск по ключу нужного элемента.

Допустим, нам необходимо вставить новый элемент за элементом с искомым значением.

Для этого нужно настроить вспомогательный указатель на голову списка, и перемещать его в цикле на следующий элемент до тех пор, пока не будет достигнуто искомое значение либо конец списка. Чтобы найти искомое значение, нужно сравнить ключ с информационным полем элемента списка, на который настроен вспомогательный указатель. Результат поиска элемента схематично представлен на рис. 45.

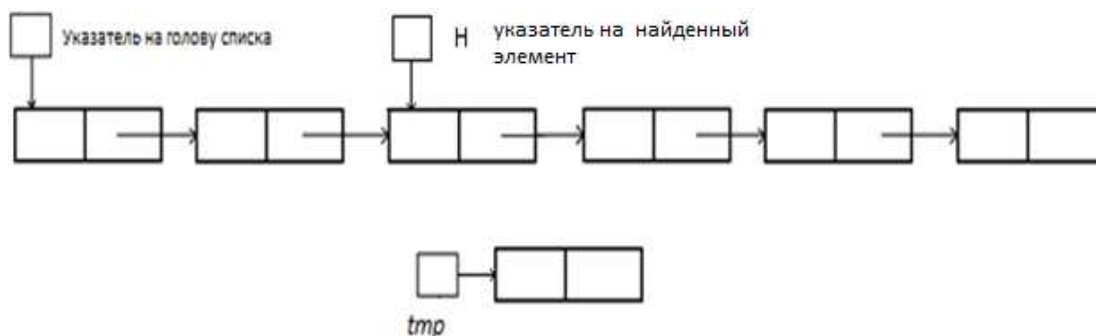


Рис. 45. Поиск элемента по ключу

```
int k;//ключ
. . .
if (H->inf==k) H=H->p;//фрагмент реализации поиска
где H – указатель на найденный элемент.
```

Далее необходимо настроить поле указатель нового элемента списка на элемент, на который указывает элемент с совпавшим ключом

```
tmp->p=H->p;
```

А поле-указатель найденного элемента настроить на новый элемент

```
H->p=tmp;
```

Результат такой перенастройки представлен на рис. 46.

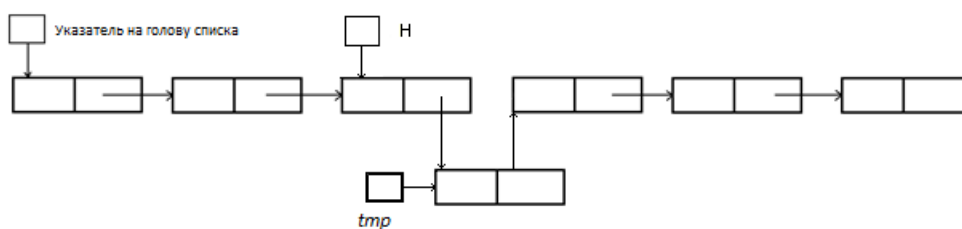


Рис. 46. Элемент вставлен в список

Если необходимо вставить новый элемент перед элементом с заданным ключом, то проверяют на совпадение с ключом элемент, следующий за элементом, на который настроен вспомогательный указатель.

Отдельно стоит рассмотреть ситуацию, когда необходимо вставить элемент перед первым. На рис. 47 представлена следующая ситуация: новый элемент уже создан, но ещё не вставлен в список.



Рис.47. Подготовка к добавлению элемента

В этом случае поле-указатель нового элемента настраивают на первый элемент, после чего перенастраивают указатель на голову списка на вновь добавленный элемент. Либо поле указатель первого элемента настраивают на новый в зависимости от организации направления списка, таким образом, если на схеме стрелки будут направлены в другую сторону, это означает, что поля-указатели настроены на предыдущий элемент, а не на следующий. На рис. 48 схематично представлен результат добавления элемента в голову списка.



Рис. 48. Элемент добавлен в голову списка

### ***Исключение элемента из списка***

Прежде чем освободить память, занимаемую элементом необходимо выполнить перенастройку указателей.

*Алгоритм удаления первого элемента:*

- настроить вспомогательный указатель на голову (рис.49);

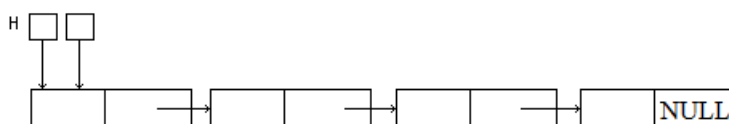


Рис. 49. Дополнительный указатель настроен на первый элемент динамического списка

– перенастроить указатель на голову на следующий элемент (рис. 50);

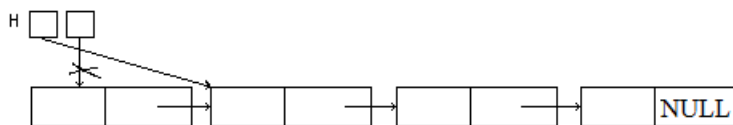


Рис. 50. Элемент удалён из головы динамического списка

– освободить память, занимаемую первым элементом.

### **Удаление элемента по ключу**

Для реализации этой задачи необходимо два вспомогательных указателя. Указатель на удаляемый элемент и указатель на предыдущий элемент.

Поле-указатель элемента, предшествующего удаляемому, настраиваем на элемент, следующий за удаляемым, после чего освобождаем память, занимаемую удаляемым из списка элементом.

Линейный список, организованный по принципу: «первый вошёл, первый вышел» (*FIFO – first in, first out*) называют **очередью**.

Для наглядности целесообразно представить очередь в бытовом понимании, например, очередь в магазин, кто первый в очередь встал, тот первым товар и получит. В случае, динамического списка элемент, который был добавлен в такой список первым, будет извлечён из очереди первым.

Для реализации такого подхода необходимо объявить указатель на голову (вершину) списка и указатель на хвост (последний элемент) очереди для возможности добавления элементов в конец очереди. Эти два указателя, можно объединить в одной переменной – структуре. Эта структура будет не того же типа, что и элементы списка, это другой пользовательский тип данных, который должен быть описан отдельно после описания структуры - элемента списка. Например, так:

```
struct Q
{
    elem* head;
    elem* tail;
};
```

Если планируется работа не более, чем с одним списком, поля `head` и `tail` можно объявить отдельными переменными вне структуры.

Для добавления элемента в очередь потребуется дополнительный указатель.

```
elem* tmp;
```

```
tmp = new elem;
```

*Алгоритм добавления элемента в очередь:*

– захватить память под новый элемент списка

```
elem* tmp;
```

```
tmp = new elem;
```

– Заполнить его поля

```
tmp->inf = a;
```

– если это первый элемент списка (т.е. указатель на голову ни на что не указывает `head == NULL`) настраиваем его на новый элемент

```
head = tmp;
```

этот же элемент будет последним, в силу единственности, т.е. указатель на хвост нужно также настроить на него;

– если элемент не первый, то указатель на голову не трогаем.

Перенастраиваем указатель на хвост на вновь захваченный элемент

```
tail = tmp;
```

– предварительно настраиваем на него же поле-указатель предыдущего элемента очереди

```
tail->p = tmp;
```

Линейный список, организованный по принципу «последний вошёл, первый вышел». (*LIFO – last in, first out*) – называют **стеком**.



Для понимания структуры такой организации данных и способов работы с ними стек представляют в виде узкой трубки, закрытой с одного конца и заполненной теннисными мячами. Как показано на рис. 51.



Рис. 51. Наглядное представление стека

Из рис. 51 видно, что в противоположность очереди, элемент, добавленный в стек первым, будет извлечён из него последним.

*Добавление элемента в стек:*

– на первом шаге захватываем память под элемент и заполняем его информационное поле

```
elem* tmp;  
tmp = new elem;  
tmp->inf = a;
```

– дальнейшая настройка зависит от того, первый это элемент или в стеке уже есть значения

```
(head == NULL);
```

– если элемент первый, указатель на вершину стека настраиваем на него

```
head = tmp;  
tmp->p = NULL;
```

– если в стеке уже есть значения, производим перенастройку указателей, таким образом, чтобы не потерять связь между элементами и при этом, новый элемент стал вершиной стека

```
tmp->p = head;  
head = tmp;
```

При написании функций работы с динамическими списками, параметры–указатели передают через ссылки, если указатель передать по значению, изменённое значение не сохранится после выхода из функции.

## **Лабораторная работа № 9**

### **ЛИНЕЙНЫЕ ДИНАМИЧЕСКИЕ СПИСКИ**

#### ***Цель работы***

Закрепление навыков разработки функций на языке программирования C++, в том числе навыков передачи параметров в функции, возвращения значения из функции, использование этого значения в точке вызова, работы с динамической памятью.

Получение навыков организации динамических структур данных и дальнейшей работы с ними на языке программирования C++.

#### ***Постановка задачи***

Реализовать работу с линейными динамическими списками (организованными по принципу стека и очереди) для хранения целочисленных данных.

#### ***Указания к выполнению лабораторной работы***

Написать функции добавления и удаления элементов в стек, функции добавления и удаления элементов из очереди, в функциях предусмотреть проверку на пустоту списка.

При этом, в функциях добавления элемента, если список пуст, необходимо создать первый элемент. Если список пуст и при этом вызвана функция извлечения элемента, нужно выдать соответствующее сообщение.

Создать и проинициализировать целочисленный массив.

В цикле перебора элементов, добавить элементы в стек и очередь. Каждый элемент должен быть добавлен в обе структуры.

После того как, все элементы добавлены, извлечь и вывести на экран элементы стека и очереди.

### ***Вопросы для самоконтроля, подготовки к выполнению и защите лабораторной работы***

- 1) В каком случае структуру данных называют динамической?
- 2) Какие динамические структуры называют линейными?
- 3) Приведите примеры линейных динамических структур?
- 4) Каким образом должен быть описан элемент динамической структуры данных на языке программирования C++?
- 5) Назовите правило для организации линейной динамической структуры данных – очередь.
- 6) Назовите правило для организации линейной динамической структуры данных – стек.
- 7) Перечислите этапы добавления элемента в линейную динамическую структуру.
- 8) Перечислите этапы удаления элемента из линейной динамической структуры.
- 9) Каким образом нужно передать указатель на голову и хвост очереди в функцию добавления элемента?
- 10) Почему именно этим способом?

### ***Задания для самостоятельной работы***

- 1) Напишите функцию добавления элемента в очередь.
- 2) Напишите функцию добавления элемента в стек.
- 3) Напишите функцию проверки линейного динамического списка на пустоту.
- 4) Напишите функцию извлечения элементов из линейного динамического списка.
- 5) Напишите функцию поиска элемента в списке по ключу.
- 6) Напишите функцию удаления по ключу элемента из линейного динамического списка.
- 7) Напишите функцию вставки по ключу элемента в линейный динамический список.
- 8) Напишите функцию создания/добавления элемента в однонаправленный кольцевой список.

9) Напишите функцию создания/добавления элемента в двунаправленный кольцевой список.

10) Напишите функцию удаления элемента по ключу из двунаправленного кольцевого динамического списка.

11) Элементы целочисленного массива записать в очередь. Написать функцию извлечения элементов из очереди до тех пор, пока первый элемент очереди не станет чётным.

12) Даны две очереди, содержащие целые числа. Объединить очереди в одну, таким образом, чтобы первой в результирующей очереди оказалась та, сумма элементов, которой больше.

13) Даны две непустые очереди, одинаковой длины. Объединить очереди в одну, в которой элементы исходных очередей чередуются.

14) Даны две непустые очереди. Элементы каждой из очередей упорядочены по возрастанию. Объединить очереди в одну с сохранением упорядоченности элементов.

15) Пусть имеется файл действительных чисел и некоторое число  $A$ . Используя очередь, вывести на экран сначала все элементы, меньшие числа  $A$ , а затем все остальные элементы.

## ЗАКЛЮЧЕНИЕ

Получение современного образования невозможно без изучения основ алгоритмизации и программирования, получения навыков в области разработки, отладки, тестирования и модификации программного обеспечения.

Освоение курса «Основы алгоритмизации и программирования» даёт студенту возможность получения навыков самостоятельной разработки алгоритмических решений, представления решения в виде блок-схемы, написания кода на языке программирования высокого уровня, использования и отладки для дальнейшей работы ранее написанного кода, в том числе и кода, написанного другими разработчиками.

Полученные знания и навыки могут быть применены будущими специалистами при решении широкого спектра профессиональных задач.

## РЕКОМЕНДАТЕЛЬНЫЙ БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Павловская Т. А., С/С++. Программирование на языке высокого уровня. Учебник для вузов. — СПб.: Питер, 2009. — 432 с: ил. ISBN 978-5-91180-174-8

2. Страуструп, Б. Программирование. Принципы и практика с использованием С++ / Б. Страуструп. – М. : Вильямс, 2016. – 1328 с. – ISBN 978-5-8459-1949-6. 5.

3. Шилдт. Г. Полный справочник по С++ : пер. с англ. / Г. Шилдт. – 4-е изд., стер. – М. : Вильямс, 2015. – 800 с. – ISBN 978-5-8459-2047-8.

4. 1. Воронова, Л. М. Типовые алгоритмические структуры для вычислений : учеб. пособие / Л. М. Воронова ; Владим. гос. ун-т. – 2-е изд., перераб. и доп. – Владимир : Ред.-издат. комплекс ВлГУ, 2004. – 96 с. – ISBN 5-89368-503-2.

5. Прата, С. Язык программирования С++. Лекции и упражнения : пер. с англ. / С. Прата. – М. : Вильямс, 2017. – 1248 с. – ISBN 978-5-8459-2048-5.

## **ПРИЛОЖЕНИЕ**

### **Образец титульного листа отчёта по лабораторной работе**

Министерство науки и высшего образования Российской Федерации  
Федеральное государственное бюджетное образовательное учреждение  
высшего образования  
«Владимирский государственный университет  
имени Александра Григорьевича и Николая Григорьевича Столетовых»  
(ВлГУ)

Кафедра ФиПМ

### **ЛАБОРАТОРНАЯ РАБОТА № 1**

по дисциплине «Основы алгоритмизации и программирования»  
на тему: «Основные алгоритмические структуры»

Выполнил:  
ст. группы ЛТ-123  
Иванов И.И.  
Принял:  
ст. преподаватель  
каф. ФиПМ  
Шишкина М. В.

Владимир 2023

*Учебное электронное издание*

ШИШКИНА Мария Викторовна

ОСНОВЫ АЛГОРИТМИЗАЦИИ И ПРОГРАММИРОВАНИЯ

Практикум

*Издается в авторской редакции*

**Системные требования:** Intel от 1,3 ГГц; Windows XP/7/8/10; Adobe Reader;  
дисковод CD-ROM.

**Тираж 25 экз.**

Владимирский государственный университет  
имени Александра Григорьевича и Николая Григорьевича Столетовых  
Изд-во ВлГУ  
rio.vlgu@yandex.ru

Институт прикладной математики, физики и информатики  
кафедра физики и прикладной математики  
familyshishka@mail.ru