Огнёва М.В., Кудрина Е.В.

ОСНОВЫ ПРОГРАММИРОВАНИЯ НА ЯЗЫКЕ C++

ЧАСТЬ 2

Огнёва М.В., Кудрина Е.В.

ОСНОВЫ ПРОГРАММИРОВАНИЯ НА ЯЗЫКЕ C++

ЧАСТЬ 2

ООО Издательский Центр «Наука»
2009

УДК 681.3.026(076.1)
ББК 32.973-01я73
О38

Огнёва М.В., Кудрина Е.В.
О38 *Основы программирования на языке* С++: Учеб. пособие в 2 ч. Часть 2. - Саратов: ООО Издательский Центр "Наука", 2009. - 100 с.
ISBN 978-5-91272-780-1

Данное пособие представляет собой учебно-методическую разработку по изучению основ программирования на языке С++. Пособие состоит из двух частей. Вторая часть пособия содержит 8 разделов, в которых рассматриваются технологии программирования, представление строк в стиле С и С++, разработка рекурсивных функций, перегрузка функций, разработка функций-шаблонов, организация файлового ввода/вывода данных, тип данных структура, алгоритмы внутренней сортировки, работа с классом-контейнером вектор и механизм генерации и обработки исключений. Каждый раздел пособия содержит: теоретический материал, примеры решения типовых задач и набор упражнений, предназначенных для закрепления материала.

Пособие предназначено для студентов естественно-научных факультетов, изучающих язык С++ в рамках дисциплин компьютерного цикла. Мы надеемся, что пособие окажется полезным и для преподавателей дисциплин компьютерного цикла при подготовке и проведении соответствующих занятий.

Рецензенты:

Федорова А.Г., кандидат физико-математических наук, доцент кафедры информатики и программирования, декан факультета компьютерных наук и информационных технологий Саратовского государственного университета им. Н.Г. Чернышевского

Кондратов Д. В., кандидат физико-математических наук, доцент кафедры математики и статистики, проректор по информатизации Поволжской академии государственной службы им. П.А. Столыпина

УДК 681.3.026(076.1)
ББК 32.973-01я73

Работа издана в авторской редакции на правах рукописи

ISBN 978-5-91272-780-1

Данное пособие представляет собой вторую часть учебно-методической разработки по изучению основ программирования на языке C++. Вторая часть данного пособия содержит 8 разделов.

В первом разделе пособия дается краткий обзор технологий программирования, а также вводятся основные понятия объектно-ориентированного программирования.

Во втором разделе рассматриваются два способа представления строк – в с стиле языка C и в виде объектов класса string, а также основные приемы их обработки.

В третьем разделе мы возвращаемся к изучению функций, в частности, рассматриваются различные виды рекурсивных функций, возможности перегрузки функций, а также разработка функций-шаблонов.

В четвертом разделе рассматривается организация файлового ввода/вывода данных.

В пятом разделе рассматривается тип данных структура, а также связи между структурами и классами в языке C++.

Шестой раздел посвящен алгоритмам внутренней сортировки, а именно, сортировке методом «пузырька», сортировке вставками, сортировке посредством выбора и сортировке Шелла.

В седьмом разделе дается общее представление о стандартной библиотеке шаблонов STL и подробно рассматривается класс-контейнер вектор, а также работа с ним через итераторы и стандартные алгоритмы STL.

В последнем разделе рассматривается механизм обработки исключений и его применение при решении практических задач.

Все разделы данного пособия имеют следующую структуру: теоретическая часть, примеры программ с подробными пояснениями, набор упражнений, рассчитанный на группу из 20 человек. Некоторые разделы завершаются заданиями, для решения которых от читателей потребуется самостоятельное изучение дополнительной литературы. Ссылки на литературу, список которой приведен в конце пособия, поможет более подробно разобраться в том или ином разделе пособия или выполнить самостоятельную работу.

Материалы этого пособия использовались на лекционных и практических занятиях со студентами факультета компьютерных наук и информационных технологий и механико-математического факультета Саратовского государственного университета им. Н.Г. Чернышевского.

Авторы благодарят за помощь, поддержку и критические замечания Федорову Антонину Гавриловну и всех сотрудников кафедры информатики и программирования и кафедры математической кибернетики и компьютерных наук Саратовского государственного университета им. Н.Г. Чернышевского, принимавших участие в апробации материала, изложенного в данном пособии.

1. ТЕХНОЛОГИИ ПРОГРАММИРОВАНИЯ

1.1. Эволюция технологий программирования

В окончательном виде любая программа представляет собой набор инструкций процессора. Все что написано на любом языке программирования - это более удобная для человека запись набора инструкций, облегчающая написание, отладку и последующую модификацию программ.

Первые программы создавались посредством переключателей на передней панели компьютера. Такой способ подходил только для очень небольших программ. Потом программы стали писать на языке машинных команд. Затем был изобретен язык ассемблер, который позволил писать более длинные программы. Ассемблер относится к языкам низкого уровня, но не потому, что программы, разработанные на данном языке «низкого» качества, а потому, что для написания самой простой программы программисту требовалось знать команды конкретного типа процессора и напрямую обращаться к данным, размещенным в его регистрах.

Развитие вычислительной техники вело к быстрой смене типов и моделей процессоров, поэтому стало остро необходимо обеспечивать аппаратную переносимость программ. Это привело к появлению языков программирования высокого уровня, первым из которых был Фортран. Его существенным отличием стало то, что программа разрабатывалась на языке, «схожим» с естественным языком, а для перевода программы в машинный код – в набор инструкций для конкретного типа процессора – стал использоваться транслятор. Теперь, чтобы перенести программу с одной аппаратной платформы на другую, следовало «перетранслировать» исходный код программы с помощью соответствующей версии транслятора.

С ростом объема программ становилось невозможным удерживать в памяти все детали. Стало необходимо структурировать информацию, выделять главное и отбрасывать несущественное, т.е. повышать степени абстракции программы, что привело к появлению структурного программирования.

Первым шагом в данном направлении стало использование подпрограмм. Использование подпрограмм позволяет после их разработки и отладки отвлечься от деталей реализации. При этом для вызова подпрограммы требуется знать только ее интерфейс, который, если не используются глобальные переменные, полностью определяется заголовком подпрограммы.

Вторым шагом в данном направлении стала возможность создания собственных типов данных, позволяющих структурировать и группировать информацию, представлять ее в более естественном виде. Естественно, что для работы с собственными типами данных разрабатываются специальные подпрограммы.

Объединение в модули описаний собственных типов данных и подпрограмм для работы с ними стало следующим шагом в развитии структурного программирования. Создаваемые модули стали помещаться в библиотеку подпрограмм. Таким образом, во все языках программирования, в том числе, и в языке C++, появилась обширная библиотека стандартных подпрограмм. Следует

отметить, что каждый модуль стандартной библиотеки реализует некоторую функциональность (возможность) языка, которая может использоваться другими программистами. Если в языке не хватает той или иной функциональности, то создается очередной модуль и помещается в библиотеку подпрограмм.

Структурный подход в программировании и использование готовых библиотек позволило успешно создавать достаточно крупные проекты, а также уменьшить время разработки проектов и облегчить возможность их модификации. Однако сложность программного обеспечения продолжала возрастать, и требовались все более сложные средства ее преодоления. Идеи структурного программирования получили свое дальнейшее развитие в объектно-ориентированном программировании (ООП) – технологии, позволяющей достичь простоты структуры и управляемости очень крупных программных систем.

Основные принципы ООП были разработаны еще в языках Simula-67 и Smalltalk, но в то время они не получили широкого применения из-за трудностей освоения и низкой эффективности. В C++ эти принципы реализованы эффективно и непротиворечиво, что и явилось основой успешного распространения этого языка и внедрения подобных средств в другие языки программирования.

Следует отметить, что развитие технологий программирования не остановилось на ООП. С развитием Windows-технологий связано появление компонентного подхода в программировании, с развитием Internet-технологий - появление технологий Web-программирования. Одним из современных достижений в области программирования стало развитие параллельного программирования. Однако неоспоримым остается то, что именно развитие принципов ООП дало толчок к появлению этих новых технологий.

1.2. Язык C++ и ООП

В 1972 Денис Ритчи разработал язык С - первый язык, в котором удачно сочетались мощь, элегантность, гибкость и выразительность структурного программирования. В результате, С стал одним из популярных языком программирования 70-х годов.

Язык C++ начал разрабатывать Бьерн Страуструп в 1979 году. Первоначально он был назван "С с классами", но в 1983 году это имя было изменено на C++. C++ полностью включает элементы языка С, а большинство внесенных дополнений предназначены для поддержки ООП.

ООП – это не просто набор новых средств, добавленных в язык С, это новая парадигма программирования. Термин «парадигма» означает набор теорий, стандартов и методов, совокупность которых определяет способ организации знаний – иными словами, способ видения мира. В программировании этот термин используется для определения модели вычислений, то есть способа структурирования информации, организации данных и вычислений.

Парадигма ООП основывается на введении понятия «класс», который является естественным продолжением модульности программирования. В классе структуры данных и функции их обработки объединяются. При этом класс используется только через свой интерфейс, а детали реализации для пользователя класса несущественны. Например, в первой части данного пособия мы уже

неоднократно использовали стандартный класс языка C++ для организации потокового ввода/вывода данных (класс iostream) и операции, определенные в нем, (<< и >>) не вникая в детали их реализации.

Следует отметить, что в программном понимании класс является типом данных, определяемым пользователем. В классе задаются свойства и поведение какого-либо предмета или процесса в виде членов класса – полей данных (членов-данных) и функций (членов-функций) для работы с ними. Создаваемый тип данных обладает практически теми же свойствами, что и стандартные типы. Напомним, что тип данных определяет внутреннее представление данных в памяти компьютера, множество значений, которые могут принимать величины этого типа, а также операции и функции, применяемые к этим величинам. Все это можно задать и в классе.

Существенным отличием классов является то, что они отражают строение объектов реального мира. В реальном мире можно, например, управлять автомобилем, не имея представления о принципе внутреннего сгорания и устройстве двигателя. Аналогично, зная только интерфейс класса – заголовки его функций, можно управлять его данными.

Конкретные величины типа данных «класс» называются экземплярами класса или объектами. Сознание объектов класса и присваивание им начальных значений выполняется с помощью специальных членов-функций – конструкторов. Объекты взаимодействуют между собой, посылая и получая сообщения. Сообщение – это запрос на выполнение действия, содержащий набор необходимых параметров. Механизм сообщений реализуется с помощью вызовов соответствующих членов-функций классов. Таким образом, ООП реализует «событийно-управляемую модель», в которой данные активны и управляют вызовом того или иного фрагмента программного кода.

Основными принципами ООП являются инкапсуляция, наследование и полиморфизм. Рассмотрим данные принципы более подробно.

Инкапсуляция – это объединение данных и функций их обработки в сочетании с сокрытием ненужной для использования этих данных информацией. Инкапсуляция повышает степень абстракции программы – для использования в программе какого-то класса не требуется знаний о его реализации, достаточно знать только его интерфейс. Это позволяет изменить реализацию класса, не затрагивая саму программу, при условии, что интерфейс класса останется прежним.

Наследование – это возможность создания иерархии классов, когда потомки наследуют все свойства своих предков, могут их изменять и добавлять новые. При этом свойства повторно не описываются, что сокращает объем программы. Иерархия классов представляется в виде древовидной структуры, в которой более общие классы располагаются ближе к корню, а более специализированные – на ветвях или листьях. В данном пособии мы подробно рассмотрим иерархию классов, отвечающих за организацию потокового ввода/вывода данных.

Полиморфизм – это возможность использования в различных классах иерархии одного имени для обозначения сходных по смыслу действий и гибко выбирать требуемое действие во время выполнения программы. Понятие полиморфизма широко используется в C++. Простым его примером может служить рассмотренная в первой части книги перегрузка функций библиотеки math (или

cmath), когда из нескольких вариантов нужной функции, например, функции pow, выбирается наиболее подходящая функция в соответствии с типами передаваемых параметров. Другой пример – использование шаблонов функций (механизм использования шаблонов будет рассмотрен в разделе 3.3), когда один и тот же код видоизменяется в соответствии с типом данных, переданных в качестве параметров. Однако чаще всего понятие полиморфизма связано с механизмом использования виртуальных функций (см. часть 3 и дополнительную литературу).

В данном пособии мы не будем затрагивать вопросы, связанные с разработкой собственных классов, однако нам предстоит научиться использовать такие стандартные классы языка C++ как string (используется для обработки строковых типов данных), fstream (используется для организации файловых потоков данных) и vector (используется для динамических «массивов» данных).

2. СТРОКИ

Строка – это последовательность символов определенной длины. Существуют два типа данных, которые используются для представления строк. Первый способ основывается на том, что строка представляет собой массив базового типа char, конец которого отмечается нулевым символом '\0'. Это старый способ представления строк, унаследованный языком C++ от языка C. Строки данного типа, называемые *строками C*, все еще широко используются. Например, строковые константы в кавычках, такие как "Hello", реализуются в C++ как строки C. Кроме того, данный тип используется во многих стандартных библиотеках.

Второй способ основывается на том, что строка представляет собой объект класса string. Это современный способ представления строк, который входит в стандарт ANSI/ISO языка C++. Класс string позволяет обрабатывать строки посредством стандартных операторов C++, использовать их в составе стандартных выражений. Кроме того, работать с классом string гораздо безопаснее, чем с массивом символов, т.к. в классе string контролируются границы массива.

В данном пособии рассмотрим оба способа представления строк.

Замечание. Для работы со строками в стиле C нужно подключить заголовочный файл *cstring*, а для работы со строковым классом нужно подключить заголовочный файл *string*.

2.2. Работа со строками в виде массивов символов

Описание и инициализация

Рассмотрим представление строк в виде массивов типа char. Например, строку "Привет" можно представить как массив из семи индексированных переменных – шести букв слова "Привет" и одного дополнительного нулевого символа '\0', служащего маркером конца строки. Символ '\0' называется *нуль-символом* или *нулевым символом*. Хотя нуль-символ и записывается в виде двух символов, на самом деле он считается одним символом, так же как и символ переноса строки на новую строчку – '\n'. Использование нуль-символа позволяет обрабатывать массивы посимвольно, корректно определяя конец строки.

Формат объявления строки в виде массива символов:

```
char <имя строки> [n+1];
```

где n максимальное количество символов в строке. Единица прибавляется для того, чтобы строка могла вместить не только требуемое количество символов в строке, но и нуль-символ. При этом строка может быть заполнена не до конца, т.е. символов в ней может быть меньше, чем указано в ее описании. Например, объявление:

```
char s[20];
```

говорит о том, что в строковую переменную s можно записать не более чем 19 символов.

Строковую переменную можно инициализировать при объявлении, при этом не обязательно указывать ее размер. В этом случае размер устанавливается автоматически и будет на 1 больше количества символов, заключенных в кавычки. Например, следующая инициализация:

```
char a[10]= "Привет";
char b[]="привет";
```

говорит о том, что в строковой переменной a заполнено только 6 символов из 9. А размерность переменной b равна 7 (6 символов строки "привет" плюс нуль-символ)

Если же указанный в скобках размер массива символов окажется меньше, чем количество символов в кавычках, то программа выдаст сообщение об ошибке. Например, ошибочным будет описание:

```
char a[5]= "Привет";
```

Ввод-вывод

Вывод строк осуществляется обычным образом, т.е с использованием объекта cout и операции <<. Например, вывести на экран строковую переменную a, описанную и инициализированную выше, можно следующим образом:

```
cout <<a;
```

Ввод строк можно осуществлять, используя объект cin и операцию >>. При этом символы строки будут считываться до тех пор, пока не встретится символ пробела, табуляции или перевода строки, после чего ввод прекратится.

Для того чтобы ввести строку целиком (до символа перевода строки), необходимо использовать функцию getline объекта cin. Функция cin.getline(s,n) осуществляет ввод строки s, а параметр n задает максимальное количество символов в этой строке. Если вы введете больше символов, чем задано параметром n, то лишние символы будут проигнорированы.

Например, ввести строку s, объявленную ранее, можно следующим образом:

```
cin.getline(s,20);
```

Замечания

1. При указании количества символов в функции getline не забывайте про нуль-символ.
2. Если необходимо считать несколько строк, то можно использовать функцию getline с тремя аргументами, где третьим аргументом будет символ, на котором заканчивается считывание.

Основные операции

Обратиться к символу строки можно так же как и к элементу обычного массива, указав имя строковой переменной и, в квадратных скобках, индекс требуемого символа. При этом нумерация символов в строке, так же как и в обычном массиве, начинается с 0. Рассмотрим на примере, как можно вывести на экран содержимое данной строки, обращаясь к ней посимвольно:

```
char str[20];
cin.getline(str,20);
int i=0;
while (str[i]!='\0')
  cout <<str[i++]<<'\t';
```

При работе с массивом символов надо следить, чтобы символ '\0' не был заменен каким-нибудь другим значением, т.к. при отсутствии этого символа массив перестанет вести себя как строковая переменная. Хотя программа сможет продолжить работу, но результаты этой работы будут непредсказуемыми так как произойдет выход за пределы массива и будет обрабатываться какой-то фрагмент памяти, не имеющий отношения к нашей строке.

Строки в стиле С отличаются от переменных других типов данных, и многие операции языка С++ к ним неприменимы. Так, нарпимер, нельзя использовать строковую переменную в стиле С в операторе присваивания (хотя при объявлении строковой переменой можно пользоваться знаком равенства для ее инициализации, но больше нигде в программе использовать операцию присваивания не разрешается).

Чтобы присвоить значение строковой переменной в стиле С, можно воспользоваться стандартной функцией strcpy:

```
strcpy(s, "Привет"); //присваивает переменной s значение строки "Привет"
```

Данная версия функции strcpy не проверяет, превышает ли размер строки размер строковой переменной. Более безопасной версией присваивания является функция strncpy, которая, к сожалению, существует не во всех реализациях Си++. Например, обращение к функции strncpy следующим образом:

```
strncpy(s1, s2, 10);
```

копирует 10 символов из строковой переменной s2 (независимо от ее длины) в строковую переменную s1. Например, если в строке s2 будет содержаться менее 10 символов, то в строку s1 скопируются только те символы, что имеются.

Проверку эквивалентности двух строковых переменных нельзя выполнять обычным способом, то есть с помощью оператора ==. Если применить его для сравнения двух строк, результат окажется неверным, и при этом даже не будет выведено сообщение об ошибке. Сравнение двух строк в стиле С на эквивалентность выполняется с помощью стандартной функции strcmp. Функция strcmp(s1,s2) возвращает отрицательное число, положительное число или 0 в зависимости от того, окажется первая из переданных ей строк меньше, больше или равной второй с точки зрения их лексикографического порядка.

Замечание. Лексикографический (словарный) порядок — это порядок сортировки на основе набора символов ASCII. Если строки состоят лишь из букв, причем либо только строчных, либо только прописных, лексикографический порядок полностью совпадает с обычным алфавитным порядком сортировки.

Если использовать возвращаемый ею результат в качестве логического выражения в операторе if или в цикле, то ненулевое значение будет преобразовано в true, а 0 — в false.

Компиляторы С++, соответствующие стандарту ANSI/ISO, поддерживают более безопасную версию функции strcmp с третьим аргументом, в котором задается максимальное количество сравниваемых символов.

Функции strcpy и strcmp располагаются в библиотеке с заголовочным файлом "cstring".

В следующей таблице приведено описание и примеры использования наиболее важных функций для работы со стоками в стиле С, которые также расположены в библиотеке с заголовочным файлом "cstring".

Функция	*Описание*	*Пример*	*Дополнительная информация*
strcpy(s1,s2)	Копирует значение строки s2 в строку s1.	char s1[10]; char s2[]="привет"; strcpy(s1,s2); после этого s1="привет"	Не проверяет, поместится ли значение s2 в s1
strncpy(s1,s2,n)	Копирует n символов строки s2 в строковую переменную s1	char s1[10]; char s2[]="привет"; strncpy(s1,s2,3); после этого s1="при"	Если значение n выбрано правильно, эта функция надежнее функции strcpy. Реализована не во всех версиях C++
strlen(s)	Возвращает целое число, равное длине строки s (нуль-символ не учитывается)	char s[]="привет"; int k; k=strlen(s); после этого k=6	
strcat(s1,s2)	Выполняет конкатенацию (слияния) строк, добавляя содержимое строки s2 в конец строки s1	char s1[20]= "Вася, "; char s2[20]= "привет"; strcat(s1,s2); после этого s1="Вася, привет" s2 осталась без изменения	Не проверяет, поместиться ли объединенная строка в s1
strncat(s1,s2,n)	Добавляет n символов из s2 в конец s1	char s1[20]= "Вася, "; char s2[20]= "привет"; strncat(s1,s2,3); после этого s1="Вася, при" s2 осталась без изменения	Если значение n выбрано правильно, то эта функция надежней strcat. Реализована не во всех версиях C++

strcmp(s1,s2)	Возвращает отрицательное число, положительное число или 0 в зависимости от того, окажется s1 меньше, больше или равной s2 с точки зрения их лексикографического порядка.	char s1[]="asd";
		char s2[]="afd";
		char s3[]="asd";
		int k=strcmp(s1,s2);
		int k1=strcmp(s1,s3);
		после этого k=1 ('s'<'f'), k1=0	

strncmp(s1,s2,n)	Возвращает отрицательное число, положительное число или 0 в зависимости от того, окажутся ли первые n символов s1 меньше, больше или равными первым n символам s2 с точки зрения их лексикографического порядка.	char s1[]="asd";
		char s2[]="afd";
		char s3[]="asd";
		int k=strcnmp(s1,s2,1);
		int k1=strcmp(s1,s3);
		после этого k=0 ("a"="a"), k1=0
			Если значение n выбрано правильно, то эта функция надежней strmcp. Реализована не во всех версиях C++

strchr(s, ch)	Функция возвращает указатель на первое вхождение символа ch в строке s, если его нет, то возвращает NULL.	char s[]="hello"; char ch='l'; char *a=strchr(s,ch); *a='L'; после этого s="heLlo";	

Преобразование строк в числа

Строка "1234" и число 1234 — это не одно и то же. Первое выражение представляет собой последовательность символов, а второе — целое число. Иногда при работе с числами бывает удобнее считывать их как строки символов, редактировать в строковом виде, а только потом преобразовывать в числа, чтобы выполнить над ними арифметические операции. Кроме того, иногда требуется выделить из строки все числа, которые там имеются, и произвести над ними какие-то арифметические действия.

Для преобразования строки в стиле C, состоящей из цифр, в число используются функции atoi, atol и atof. Для использования данных функций необходимо подключить заголовочный файл "cstdlib".

Функция atoi принимает один аргумент — строку в стиле C – и возвращает значение типа int, соответствующее представленному этой строкой числу. Так, atoi ("1234") возвращает число 1234.

Функция atol принимает один аргумент — строку в стиле C – и возвращает значение типа long, соответствующее представленному этой строкой числу. Так, atol ("1234567899") возвращает число 1234567899.

Функция atof принимает один аргумент — строку в стиле C –и возвращает значение типа float, соответствующее представленному этой строкой числу. Так, atol ("12.34") возвращает число 12.34.

При использовании этих функций нужно помнить о следующих моментах:

1) Если строка в стиле C, переданная в качестве аргумента любой из функций преобразования, не содержит число, то функция возвращает 0.
2) Если первые символы строки представляют собой число указанного типа, то это число и будет возвращаться в качестве результата, а остальные символы проигнорируются.

Например:

k=atol("d33.4") – возвратит 0
k=atol("aa") – возвратит 0
k=atol("33.4") – возвратит 33
k=atol("123a45667") – возвратит 123

2.2. Класс string

Класс string определен в библиотеке с именем "string" и отнесен к пространству имен std. Класс string позволяет работать со значениями и выражениями типа string почти так же, как со значениями простого типа данных. Так, для присваивания значения переменной типа string можно пользоваться оператором =, а для конкатенации двух строк - оператором +. Предположим, что s1, s2 и s3 являются объектами типа string, и переменные s1 и s2 содержат строковые значения. Тогда переменной s3 можно присвоить строку, состоящую из s1, в конец которой добавлена строка s2, при помощи оператора присваивания следующим образом:

```
s3=s1+s2
```

При этом нет риска, что s3 окажется слишком «маленькой» для нового значения. Если сумма значений длин строк s1 и s2 превысит вместимость переменной s3, для нее будет автоматически выделена дополнительная память.

Следует отметить, что заключенные в двойные кавычки строковые литералы являются строками С, и не относятся к типу string, но их можно использовать для инициализации переменных типа string.

Объявление и инициализация

Переменную типа string можно объявить следующим образом:

```
string <имя переменной>;
```

Например:

```
string s;
```

Поскольку у класса string имеется используемый по умолчанию конструктор, который инициализирует объект типа string пустой строкой, то данная переменная сразу будет иметь значение «пустая строка».

Кроме того, у класса string есть второй конструктор с одним аргументом:

```
string <имя переменной>(<значение>);
```

Например:

```
string s("привет");
```

При таком вызове конструктора строка инициализируется значением, указанным в скобках. При обращении к данному конструктору происходит приведение типов. Заключенная в кавычки строка "привет" является строкой С, а не значением типа string. Переменной s присваивается объект типа string, содержащий те же символы и в том же порядке, что и строка "привет", но его значение не завершается нуль-символом '\0'.

Существует альтернативный синтаксис объявления переменной типа string и вызова конструктора этого типа:

```
string s = "привет";  //эквивалентно string s("привет");
```

Ввод-вывод

Точно так же, как и в случае строк в стиле С, объекты класса string можно выводить с помощью операции <<, а вводить с помощью операции >>. Например:

```
cin >> s; //ввод строки
cout<< s; //вывод строки
```

Ввод значения в строковую переменную закончится тогда, когда встретится символ пробела, табуляции или перевода строки. Если нужно, чтобы программа прочитала всю введенную пользователем строку в переменную типа string, можно воспользоваться функцией getline. Синтаксис ее использования со строковыми объектами несколько отличается от синтаксиса, описанного для строк в стиле С. В частности, для ввода с клавиатуры вызывается функция getline (не являющаяся функцией-членом объекта cin) и объект cin передается ей в качестве аргумента:

```
getline(cin, s);
```

Данная версия функции прекращает чтение, встретив символ конца строки '\n'. Версия функции getline с тремя аргументами позволяет задать другой символ, отмечающий конец входных данных. Например, запись:

```
getline(cin, s, '#');
```

указывает, что чтение данных будет прекращено тогда, когда во входном потоке встретится знак #.

Операции над строками

Класс string поддерживает доступ к символам, как к элементам массива. Т.е. если имеется строка s, то записав s[i] можно обратиться к i-ому символу данной строки.

Замечание. Нумерация символов в строке, так же как и в массиве, начинается с 0

Допустимость данного индекса никак не контролируется, поэтому если индекс окажется большим, чем количество элементов строки, последствия будут непредсказуемыми (программа может не сообщить об ошибке, но выполнять ошибочные действия, т.е. обратиться в какой-то фрагмент памяти, не имеющий отношения к нашей строке).

Чтобы избежать этой проблемы, можно использовать функцию-член at. Функция-член at выполняет ту же задачу, что и квадратные скобки, но отличается от них:

- во-первых, синтаксисом – вместо вызова s[i] используется вызов s.at(i);
- во-вторых, она проверяет, допустим ли индекс, переданный ей в качестве аргумента. И если значение i при вызове s.at(i) оказывается недопустимым, функция выводит сообщение с указанием ошибки, содержащейся в программе.

Многие операции с объектами этого класса производить удобнее, чем со строками в стиле С. В частности, оператор ==, выполняемый над объектами класса string, возвращает значение true, если две строки содержат одинаковые символы в одинаковом порядке, и значение false в противном случае. Операторы сравнения <, >, <=, >= сравнивают строки с точки зрения их лексикографического порядка.

Кроме того, для работы со строками класса string можно использовать операторы присваивания = и +=:

```
s1=s2;      // в s1 записывается значение s2;
s1+=s2;     // в конец s1 добавляется s2.
```

Слияние строк можно производить с помощью операции +.

Длина строки

Чтобы узнать длину строки, т.е. количество символов в ней, можно воспользоваться функцией-членом size() или ее синонимом – функцией-членом length(). Например:

```
string s;                                   или      string::size_type
string::size_type len=s.size();                      len=s.length();
```

Замечание. В классе string определен вспомогательный тип size_type как синоним типа unsigned long или unsigned int, т.е. достаточно большой, чтобы содержать размер любой строки. Чтобы воспользоваться этим типом, можно применить оператор области видимости :: .

Чтобы выяснить, является ли строка пустой, можно проверить, равна ли ее длина 0 или использовать функцию-член empty(), которая возвращает значение true, если строка пуста и false в противном случае.

Выделение подстроки

Для выделения подстроки из исходной строки используется функция substr().

Функция s.substr(i,n) возвращает подстроку строки s, которая начинается с позиции i и содержит n символов.

Функция s.substr(i) возвращает подстроку строки s, которая начинается с позиции i и заканчивается последним символом строки s.

Сравнение строк

Как уже говорилось, в классе string определены все операторы, которые позволяют выяснить равенство (==) или неравенство (!=) двух строк, а также сравнивать их (<, <=, >, >=). Кроме операторов сравнения класс string предоставляет набор перегруженных версий функции compare(), которые осуществляют лексикографическое сравнение.

Пусть определены следующие типы:

```
string s,s1;
char sc[20];
```

Тогда:

1) s.compare(s1) – возвратит 0, если s и s1 равны, отрицательное число, если s<s1 и положительное, если s>s1;
2) s.compare(sc) – аналогичным образом сравнивает строку s со строкой sc в стиле C;
3) s.compare(i,n,s1) – сравнивает n символов строки s, начиная с позиции i, со строкой s1;
4) s.compare(i,n,s1,i1,n1) – сравнивает n символов строки s, начиная с позиции i, с n1 символами строки s1 начиная с позиции i1;
5) s.compare(i,n,sc,n1) – сравнивает n символов строки s, начиная с позиции i, с n1 символами строки sc в стиле C.

Поиск в строке

Класс string предоставляет 6 вариантов функции поиска. Все они возвращают либо значение типа string::size_type, которое является индексом найденного элемента, либо специальное значение string::npos, если ничего не найдено. Каждый из вариантов функции поиска существует в нескольких версиях, которые отличаются набором аргументов. В следующей таблице приведены различные варианты функции поиска:

s.find(args)	ищет первое вхождение аргумента в строку s
s.rfind(args)	ищет первое справа вхождение аргумента в строку s
s.find_first_of(args)	ищет первое вхождение любого символа аргумента в строку s
s.find_last_of(args)	ищет последнее вхождение любого символа аргумента в строку s
s.find_first_not_of(args)	ищет первое вхождение символа, который отсутствует в аргументе, в строке s
s.find_last_not_of(args)	ищет последнее вхождение символа, который отсутствует в аргументе, в строке s

В качестве аргументов функций поиска могут использоваться различные варианты:

c	в исходной строке ищется символ c
c, pos	в исходной строке ищется символ c, начиная с позиции pos
sub	в исходной строке ищется подстрока sub
sub, pos	в исходной строке ищется подстрока sub, начиная с позиции pos

Например:

```
string s="в данной строке ищем подстроку";
string sub="строк";
char c='o';
size_type i1= s.find(sub);          //i1=9
size_type i2= s.find(c);            //i2=6
size_type i3= s.rfind(sub);         //i3=24
size_type i4= s.find(c,16);         //i4=22
size_type i5= s.find_first_of(sub); //i5=6
```

Изменение строки

Для замены подстроки в строке существуют различные версии функции replace, в которых указывается часть строки, которую нужно заменить, и заменяющий ее символ или набор символов. Заменяемая часть строки задается индексом первого символа и количеством символов. Заменяющий набор символов задается строкой, массивом символов или отдельным символом. Все функции меняют строку и возвращают указатель на нее.

Например:

```
s.replace(i,n,s1);
```

заменяет n символов строки s, начиная с позиции i, на подстроку s1.

Добавить подстроку можно воспользовавшись операцией +. Например:

```
s+=s1;
```

добавляет в конец строки s значение строки s1.

Кроме этого существуют несколько версий функции append(), добавляющих подстроку, массив символов или отдельный символ в конец строки. Все функции меняют строку и возвращают указатель на нее. Например:

```
s.append(s1);      // добавляет к строке s строку s1;
s.append(n,c);     // добавляет символ с n раз;
s.append(s1,i,n);  //добавляет к строке s n символов строки s1, начиная с i-го.
```

Вставить подстроку можно с помощью функции insert(). Существует несколько версий этой функции, которые предназначены для вставки строки, массива символов или отдельного символа в указанное место данной строки. Место вставки обычно указывается первым аргументом – это индекс элемента строки, перед которым будет сделана вставка. При вставке строка раздвигается, ее длина увеличивается. Функции возвращают ссылку на измененную строку. Например:

```
s.insert(i,s1);    //вставляет в строку s строку s1, начиная с позиции i
s.insert(i, n,c);  //вставляет в строку s символ с n раз, начиная с позиции i
s.insert(i,s1,j,k); //вставляет в строку s, начиная с позиции i, k символов строки s1,
                    //начиная с j-го
```

Удалить подстроку можно с помощью функции erase. Например:

```
s.erase(i,n);      //удаляет из строки s n символов начиная с позиции i.
```

В следующей таблице содержится описание рассмотренных ранее функций-членов класса string Напомним, чтобы воспользоваться ими следует подключить заголовочный файл "string".

Обращение к функции	*Описание*
s.length()	Возвращает количество символов в строке s
s.substr(i,n)	Возвращает подстроку строки s, которая начинается с позиции i и содержит n символов
s.empty()	Возвращает true, если s является пустой строкой и false в противном случае
s.insert(i,s1)	Вставляет в строку s строку s1, начиная с позиции i
s.erase(i,n)	Удаляет из строки s n символов начиная с позиции i
s.find(s1)	Возвращает индекс первого вхождения строки s1 в строку s
s.find(s1,i)	Возвращает индекс первого вхождения строки s1 в строку s, причем поиск начинается с позиции i
s.rfind(s1)	Возвращает индекс первого справа вхождения строки s1 в строку s
s.rfind(s1,i)	Возвращает индекс первого справа вхождения строки s1 в строку s, причем поиск ведется начиная с позиции i
s.replace(i,n,s1)	Заменяет n символов строки s начиная с позиции i на подстроку s1
s.compare(s1)	Возвращает 0, если s и s1 равны, отрицательное число, если s<s1 и положительное, если s>s1

2.3 Взаимное преобразование объектов типа string и строк в стиле C

Как мы уже видели, C++ выполняет автоматическое преобразование типов, позволяющее присваивать строки в стиле C переменным типа string. Например, приведенный ниже код является корректным:

```
char s1[] = "привет";
string s2;
s2 = s1;
```

А вот использование следующего оператор недопустимо:

```
s1 = s2;    //ошибка: несовместимость типов
```

Недопустим и следующий оператор:

```
strcpy(s1, s2);    //ошибка: несовместимость типов
```

Это происходит потому, что функция strcpy не может принимать в качестве второго аргумента объект типа string, и C++ не выполняет автоматического преобразования объектов string в строки C, что является реальной проблемой.

Для получения строки в стиле C, соответствующей объекту string, нужно выполнить явное преобразование типов. Это можно сделать с использованием функции-члена c_str(). Вот как правильно произвести копирование объекта string в строку C:

```
strcpy(s1, s2.c_str());
```

2.4. Работа с отдельными символами

Приведем список функций, применимых к символьным значениям. Все они проверяют переданный им символ и возвращают значение true, если проверка пройдена и false в противном случае:

isalpha(c)	возвращает true если c - буква
islower(c)	возвращает true если c – символ нижнего регистра
isuper(c)	возвращает true если c – символ верхнего регистра
isalnum(c)	возвращает true если c – буква или цифра
isdigit(c)	возвращает true если c – цифра
isxdigit(c)	возвращает true если c – шестнадцатеричная цифра
isprint(c)	возвращает true если c – печатаемый символ, включая пробел
ispunct(c)	возвращает true если c – знак пунктуации
isspace(c)	возвращает true если c – пробельный символ (пробел, табуляция, возврат каретки, перевод страницы)
isgraph(c)	возвращает true если c – не пробел, а печатаемый символ
iscntrl(c)	возвращает true если c – управляющий символ
tolower(c)	если c прописная буква, возвращает ее эквивалент в нижнем регистре, в противном случае оставляет без изменений
toupper(c)	если c строчная буква, возвращает ее эквивалент в верхнем регистре, в противном случае оставляет без изменений

2.5. Смешанный строко-числовой ввод данных

Рассмотрим следующий пример программы:

```
#include "iostream"
#include "string"
using namespace std;
int main()
```

```
{
  int age ;
  string name, address;
  cout<<"Name: ";
  getline(cin,name);
  cout<<"Age: ";
  cin>>age;
  cout<<"Address: ";
  getline(cin,address);
  cout<< "\n"<<name<<"\t"<<age<<"\t"<<address<<endl;
}
```

Замечание. В некоторых версиях компилятора вместо *#include "имя_заголовочного_файла"* вам потребуется написать *#include < имя_заголовочного_файла >*.

В данном примере мы используем смешанный ввод данных – для ввода числовой информации применяем операцию >>, а для ввода строковых данных функцию getline. Выполнение этой программы будет выглядеть примерно так:

```
Name: Petrov
Age: 23
Address:
Petrov 23
Press any key to continue
```

Вы никогда не получите возможность ввести адрес. Проблема состоит в том, что cin читает до пробельного символа (пробела, табуляции, перехода на новую строку), оставляя сам пробельный символ в потоке. В данном случае будет прочитан возраст, а так же символ новой строки '\n', сгенерированный нажатием клавиши Enter, останется во входном потоке. Функция getline воспримет символ '\n', оставшийся во входном потоке, как пустую строку и запишет ее в переменную address. Исправить эту ошибку можно только читая символ новой строки перед чтением данных в строковую переменную address. Это можно сделать, используя функцию get() без параметров, для чтения одного символа из входного потока:

```
cin.get();
```

Преобразуем нашу программу так, чтобы все данные вводились корректно:

```
#include "iostream"
#include "string"
using namespace std;
int main()
{
  int age ;
  string name, address;
  cout<<"Name: "; getline(cin,name);
  cout<<"Age: ";  cin>>age;
  cout<<"Address: ";
  cin.get();   //считывание символа \n из входного потока
  getline(cin,address);
  cout<< "\n"<<name<<"\t"<<age<<"\t"<<address<<endl;
}
```

2.6. Примеры работы со строками

1. Дана строка и символ, выяснить, сколько раз данный символ встречается в данной строке.

Замечание. Рассмотрим несколько вариантов решения данной задачи.

а) с помощью строки в стиле С, используя простой перебор символов:

```
#include "iostream"
#include "cstring"
using namespace std;
int main()
{
  char str[20];
  char symbol;
  int k=0;
  cout << "Enter string" <<endl;
  cin.getline(str,20);
  cout << "Enter symbol" <<endl;
  cin >> symbol;
  for (unsigned int i=0;i<strlen(str);i++)
    if (str[i]==symbol) k++;
  cout <<"k=" <<k <<endl;
  return 0;
}
```

б) с помощью объекта класса string, используя простой перебор символов:

```
#include "iostream"
#include "string"
using namespace std;
int main()
{
  string str;
  char symbol;
  int k=0;
  cout << "Enter string" <<endl;
  getline(cin,str);
  cout << "Enter symbol" <<endl;
  cin >> symbol;
  for (unsigned int i=0;i<str.length();i++)
    if (str[i]==symbol) k++;
  cout <<"k=" <<k <<endl;
  return 0;
}
```

в) с помощью объекта класса string, используя функцию find:

```
#include "iostream"
#include "string"
using namespace std;
int main()
{
```

```
 string str;
 char symbol;
 int k=0;
 string::size_type pos=0;        //позиция, начиная с которой ищем символ
 cout << "Enter string" <<endl;
 getline(cin,str);
 cout << "Enter symbol" <<endl;
 cin >> symbol;
 string::size_type n=str.find(symbol,pos);  //номер первого вхождения
 while (n!=string::npos) //пока номер вхождения не равен npos,
  {                      //т.е.пока данный символ в строке есть
   k++;      //счетчик увеличиваем на 1 и начинаем искать новое вхождение
   pos=n+1; // с позиции, следующей за найденной, чтобы каждый раз
   n=str.find(symbol,pos); // не находить одно и то же
  }
 cout <<"k=" <<k <<endl;
 return 0;
}
```

2. В данной строке заменить все вхождения одного символа на другой

а)с помощью строки в стиле С, используя функцию strchr

```
#include "iostream"
#include "cstring"
using namespace std;
int main()
{
 char str[20];
 char symbol,symbol1;
 int k=0;
 cout << "Enter string" <<endl;
 cin.getline(str,20);
 cout << "Enter symbol" <<endl;
 cin >> symbol;
 cout << "Enter symbol" <<endl;
 cin >> symbol1;
 char *s=strchr(str,symbol);    //устанавливаем указатель на первое вхождение
                                //символа symbol в строке str
 while (s!=NULL)                //пока указатель не пустой
  {
   *s=symbol1;                  //заменяем данное значение на введенный символ
    s=strchr(str,symbol);       //перемещаем указатель на заданный символ в
  }                             //строке str
 cout <<"str=" <<str <<endl;
 return 0;
}
```

б)с помощью объекта класса string, используя функцию find:

```
#include "iostream"
#include "string"
using namespace std;
```

```
int main()
{
  string str;
  char symbol1,symbol2;
  cout << "Enter string" <<endl;
  getline(cin,str);
  cout << "Enter symbol 1" <<endl;
  cin >> symbol1;
  cout << "Enter symbol 2" <<endl;
  cin >> symbol2;
  string::size_type k=str.find(symbol1);
  while (k!=string::npos)
    {
      str[k]=symbol2;
      k=str.find(symbol1);
    }
  cout <<"string=" <<str <<endl;
  return 0;
}
```

3. В данной строке заменить каждую точку многоточием

Замечание. Используем следующий алгоритм: находим первое вхождение точки в строку, вставляем после нее еще две точки. Затем снова ищем первое вхождение точки в строку, но уже с позиции pos, т.е. с позиции, следующей за последней вставленной точкой.

```
#include "iostream"
#include "string"
using namespace std;
int main()
{
  string str;
  string::size_type k=0,pos=0;
  cout << "Enter string" <<endl;
  getline(cin,str);
  k=str.find(".",pos);
  while (k!=string::npos)
    {
      str.insert(k+1,"..");
      pos=k+3;
      k=str.find(".",pos);
    }
  cout <<"string=" <<str <<endl;
  return 0;
}
```

4. Удалить из заданной строки все цифры

а) используем тот факт, что все символы упорядочены, следовательно, если данный символ является цифрой, он должен находиться между символом '0' и '9'.

```
#include "iostream"
#include "string"
```

```
using namespace std;
int main()
{
 string str;
 unsigned int k=0,pos=0;
 cout << "Enter string" <<endl;
 getline(cin,str);
 while (k<str.length())
  {
   if (str[k]>='0' && str[k]<='9')
     str.erase(k,1);
     else k++;
  }
 cout <<"string=" <<str <<endl;
 return 0;
}
```

б)используем функцию isdigit, которая возвращает значение true, если ее аргумент цифра

```
#include "iostream"
#include "string"
using namespace std;
int main()
{
 string str;
 unsigned int k=0,pos=0;
 cout << "Enter string" <<endl;
 getline(cin,str);
 while (k<str.length())
  {
    if (isdigit(str[k]))
      str.erase(k,1);
      else k++;
  }
 cout <<"string=" <<str <<endl;
 return 0;
}
```

5. Дана строка, которая представляет собой слова, разделенные пробелами. Вывести на экран количество слов в строке, которые начинаются и заканчиваются одной и той же буквой.

Замечание. *Используем следующий алгоритм: находим первое вхождение пробела в строку и выделяем подстроку от первого символа до первого пробела. Получаем первое слово. Затем пересчитываем номер позиции, с которой начинаем поиск, и повторяем описанные действия.*

```
#include "iostream"
#include "string"
using namespace std;
int main()
{
 string str,slovo;
```

```
 int s=0;
 string::size_type k=0,pos=0;
 cout << "Enter string" <<endl;
 getline(cin,str);
 str=str+' ';
 k=str.find(" ",pos);
 while (k!=string::npos)
  {
    slovo=str.substr(pos,k-pos);
    if (slovo[0]==slovo[slovo.length()-1]) s++;
    pos=k+1;
    k=str.find(" ",pos);
  }
 cout <<"kolichestvo=" <<s <<endl;
 return 0;
}
```

6. Рассмотрим усложненный вариант предыдущей задачи, когда слова в строке могут разделяться пробелами и знаками препинания. Предложение заканчивается точкой.

Замечание. Используем вариант функции поиска, который ищет первый из набора символов. Еще одно отличие от предыдущей программы заключается в том, что если после слова стоит знак пунктуации, то в следующей позиции будет пробел, а не начало нового слова. А пробел тоже входит в набор искомых символов (razdel), поэтому в этом случае надо перейти на две позиции вперед (pos=k+2).

```
#include "iostream"
#include "string"
using namespace std;
int main()
{
 string str,slovo;
 int s=0;
 string::size_type k=0,pos=0;
 string razdel=",.:;!? ";
 cout << "Enter string" <<endl;
 getline(cin,str);
 k=str.find_first_of(razdel,pos);
 while (k!=string::npos)
   {
     slovo=str.substr(pos,k-pos);
      if (ispunct(str[k]))
        pos=k+2;
        else pos=k+1;
     if (slovo[0]==slovo[slovo.length()-1]) s++;
     k=str.find_first_of(razdel,pos);
   }
 cout <<"kolichestvo=" <<s <<endl;
 return 0;
}
```

2.7. Упражнения

I. Поставьте знак сравнения (>,<,==) между парами строк и обоснуйте свой ответ:

1. "Процессор"___"Процесс"
2. "Balkon"___"balkon"
3. "кошка "___"кошка"
4. "окно"___"оКно"
5. "муха"___"СЛОН"
6. "кино"___"кино"
7. " мышь"___"мышь"
8. "Иванов"___"Петров"
9. "Смирнов"___"Смирнова"

II. Простые действия со строками.

1. В данной строке вставить символ c1 после каждого вхождения символа c2.
2. Вставить после каждого вхождения подстроки str1 подстроку str2.
3. Определить, сколько в строке гласных букв (при условии, что текст записан кириллицей).
4. Выяснить, имеются ли в строке два соседствующих одинаковых символа.
5. Удвоить каждое вхождение заданной буквы в строке.
6. Удалить из строки все символы c1.
7. Удалить из строки все подстроки str1.
8. Заменить в строке все вхождения подстроки str1 на подстроку str2.
9. Определить сколько различных символов встречается в строке.
10. В строке имеются только две одинаковых буквы. Найти их.
11. Дана строка. Подсчитать количество содержащихся в ней прописных латинских букв.
12. Даны целые положительные числа N1 и N2 и строки S1 и S2. Получить из этих строк новую строку, содержащую первые N1 символов строки S1 и последние N2 символов строки S2 (в указанном порядке).
13. Дан символ C и строки S, S0. Перед каждым вхождением символа C в строку S вставить строку S0.
14. Дан символ C и строки S, S0. После каждого вхождения символа C в строку S вставить строку S0.
15. Даны строки S и S0. Найти количество вхождений строки S0 в строку S.
16. Даны строки S и S0. Удалить из строки S все подстроки, совпадающие с S0. Если совпадающих подстрок нет, то вывести строку S без изменений.
17. Дана строка, содержащая по крайней мере один символ пробела. Вывести подстроку, расположенную между первым и вторым пробелом исходной строки. Если строка содержит только один пробел, то вывести пустую строку.
18. Дана строка, содержащая по крайней мере один символ пробела. Вывести подстроку, расположенную между первым и последним пробелом исходной строки. Если строка содержит только один пробел, то вывести пустую строку.

19. Дана строка-предложение. Зашифровать ее, поместив вначале все символы, расположенные на четных позициях строки, а затем, в обратном порядке, все символы, расположенные на нечетных позициях (например, строка «Программа» превратится в «ргамамроП»).

III. Сложные действия со строками

Дано осмысленное текстовое сообщение (т.е. алфавитно-цифровая информация, разделенная пробелами и знаками препинани, в конце которого ставится точка):

1. Подсчитать, сколько раз заданное слово встречается в сообщении (при этом заданное слово не может являться частью другого слова).
2. Найти самое длинное слово в сообщении.
3. Найти самое короткое слово в сообщении.
4. Вывести на экран все слова-палиндромы, содержащиеся в заданном сообщении.
5. Подсчитать количество слов, содержащихся в сообщении.
6. Вывести на экран все слова сообщения, состоящие из n букв.
7. Вывести только те слова сообщения, которые начинаются на заданную букву.
8. Вывести только те слова сообщения, которые начинаются и оканчиваются на одну и ту же букву.
9. Вывести только те слова сообщения, которые начинаются и оканчиваются на заданную букву.
10. Вывести только те слова сообщения, которые встречаются в нем ровно один раз.
11. Вывести только те слова сообщения, которые встречаются в нем более n раз.
12. Вывести все слова сообщения, которые содержат данную букву.
13. После каждого заданного слова в сообщении поставить восклицательный знак.
14. Преобразовать сообщение так, чтобы каждое его слово начиналось с заглавной буквы.
15. Поменять слова в сообщении по принципу: первое со вторым, третье с четвертым и т.д.
16. Поменять слова в сообщении по принципу: первое с последним, второе с предпоследним и т.д.
17. Поменять слова в сообщении по принципу: первое с n/2+1-словом, второе с n/2+2-словом, i-тое с n/2+i-словом и т.д (n-число слов в предложении).
18. Удалить из сообщения все однобуквенные слова.
19. Удалить из сообщения все слова, начинающиеся с заглавной буквы.
20. Удалить из сообщения все повторяющиеся слова.

IV. Преобразование символов в числа

1. Дан текст, содержащий цифры. Вывести на экран все имеющиеся в нем цифры.
2. Дан текст, содержащий цифры. Вывести на экран только нечетные цифры.
3. Дан текст, содержащий цифры. Вывести на экран количество цифр в нем.
4. Дан текст, содержащий цифры. Вывести на экран их сумму.
5. Дан текст, содержащий цифры. Вывести на экран наибольшую цифру.
6. Дан текст, содержащий цифры. Найти наибольшее количество идущих подряд цифр.
7. Дан текст, содержащий цифры. Заменить все нечетные цифры наименьшей цифрой, содержащейся в данном тексте.
8. Дан текст, имеющий вид: d1 + d2 + ... + dn, где di — цифры. Вычислить записанную в тексте сумму.

9. Дан текст, имеющий вид: d1 - d2 + d3 - ..., где di — цифры. Вычислить значение данного выражения.
10. Дан текст, имеющий вид: d1 ± d2 ± ... ± dn, где di — цифры. Вычислить значение данного выражения.
11. Дан текст, содержащий цифры. Найти наибольшее количество идущих подряд цифр.
12. Дан текст. Определить, является ли он правильной десятичной записью целого числа.
13. Дан текст, представляющий собой десятичную запись целого числа. Вычислить сумму цифр этого числа.
14. Дан текст, содержащий целые числа. Вывести на экран все имеющиеся в нем числа.
15. Дан текст, содержащий целые числа. Вывести на экран только четные числа.
16. Дан текст, содержащий целые числа. Вывести на экран количество чисел в нем.
17. Дан текст, содержащий целые числа. Вывести на экран сумму нечетных чисел.
18. Дан текст, содержащий целые числа. Вывести на экран наименьшее из имеющихся чисел.
19. Дан текст. Определить, является ли он правильной десятичной записью вещественного числа.
20. Дан текст, содержащий вещественные числа. Вывести на экран все вещественные числа, содержащиеся в нем.

2.8. Самостоятельная работа

Задача 1. Известны фамилия, имя и отчество пользователя. Найти его код личности. Правило получения кода личности: каждой букве ставится в соответствие число – порядковый номер буквы в алфавите. Эти числа складываются. Если полученная сумма не является однозначным числом, то цифры числа снова складываются и так до тех пор, пока не будет получено однозначное число. Например:

Исходные данные: Александр Сергеевич Пушкин

Код личности: (1+13+6+12+19+1+15+5+18)+(19+6+18+4+6+6+3+10+25)+(17+21+26+12+10+15)=288 ⇒ 2+8+8=18 ⇒ 1+8=9

Задача 2. В шифре Цезаря алфавит размещается на круге по часовой стрелке. За последней буквой алфавита идет первая буква алфавита, т.е. после буквы «я» идет буква «а». При шифровании текста буквы заменяются другими буквами, отстоящими по кругу на заданное количество позиций (сдвиг) дальше по часовой стрелке. Например, если сдвиг равен 3, то буква «а» заменяется на букву «г», буква «б» на букву «д», а буква «я» на букву «в».

Зашифровать сообщение, используя шифр Цезаря со сдвигом k.

Например, дан алфавит {a, b, c}, с помощью которого записано слово *abaccc*. Использовуя шифр Цезаря со сдвигом 2 мы получим слово *cacbbb*.

3. ФУНКЦИИ В С++

Напомним, что в разделе «2. Функции в С++» первой части данного пособия было введено понятие функции, а также рассмотрены различные способы описания, определения и вызовы функций. В данном разделе мы продолжим изучение данной темы.

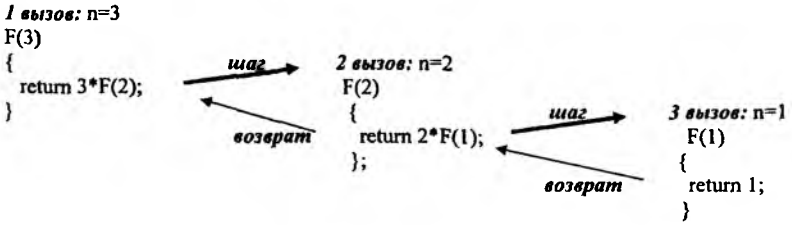
3.1. Рекурсивные функции

Рекурсивной называют функцию, если она вызывает сама себя в качестве вспомогательной. В основе рекурсивной функции лежит так называемое «рекурсивное определение» какого-либо понятия. Классическим примером рекурсивной функции является функция, вычисляющая факториал.

Из курса математики известно, что 0!=1!=1, n!=1*2*3…*n. С другой стороны, n!=(n-1)!*n. Таким образом, известны два значения параметра n, а именно, n=0 и n=1, при которых мы без каких-либо дополнительных вычислений можем определить значение факториала. Во всех остальных случаях, то есть для n>1, значение факториала может быть вычислено через значение факториала для n-1. Таким образом, рекурсивная функция будет иметь вид:

```
#include "iostream"
using namespace std;
unsigned long F(short  n)        //рекурсивный метод
{
  if (n==0 || n==1)
       return 1;          //нерекурсивная ветвь
  //шаг рекурсии - повторный вызов функции с другим параметром
  else return n*F(n-1);
}
int main()
{
  short n;
  cout <<"n= "; cin >>n;
  unsigned long f=F(n);  //нерекурсивный вызов функции F
  cout <<n<<"!="<< f<<endl;
  return 0;
}
```

Рассмотрим работу описанной выше рекурсивной функции для n=3.



Первый вызов функции F осуществляется из функции main. В нашем случае это происходит во время выполнения оператора присваивания f=F(3). Этап вхождения в рекурсию обозначим жирными стрелками. Он продолжается до тех пор, пока значение переменной n не становится равной 1. После этого начинается выход из рекурсии (тонкие стрелки). В результате вычислений получается, что F(3)=3*2*1.

Рассмотренный вид рекурсии называют прямой. Метод с прямой рекурсией обычно содержит следующую структуру:

```
if (<условие>)
  <оператор>;
  else <вызов данной функции с другими параметрами>;
```

В качестве <условия> обычно записываются некоторые граничные случаи параметров, передаваемых рекурсивной функции, при которых результат её работы заранее известен. Далее следует простой оператор или блок, а в ветви else происходит рекурсивный вызов данной функции с другими параметрами.

Что необходимо знать для реализации рекурсивного процесса? Со входом в рекурсию осуществляется вызов функции, а для выхода необходимо помнить точку возврата, т.е. то место программы, откуда мы пришли и куда нам нужно будет возвратиться после завершения выполнения функции. Место хранения точек возврата называется стеком вызовов, и для него выделяется определенная область оперативной памяти. В стеке запоминаются не только адреса точек возврата, но и копии значений всех параметров, по которым восстанавливается при возврате вызывающая функция. Из-за создания копий параметров при развертывании рекурсии может произойти переполнение стека. Это является основным недостатком рекурсивной функции. Другой недостаток – это время, которое затрачивается на вызовы функции. Вместе с тем рекурсивные функции позволяют перейти к более компактной записи алгоритма.

Любую рекурсивную функцию можно преобразовать в обычную функцию. И наоборот: практически любую функцию можно преобразовать в рекурсивную, если выявить рекуррентное соотношение между вычисляемыми с помощью функции значениями.

Далее каждую задачу будем решать с использованием обычной и рекурсивной функций:

Пример 1: Найти сумму цифр числа A.

Известно, что любое натуральное число $A = a_n a_{n-1} ... a_1 a_0$, где a_n, a_{n-1}, ..., a_0 - цифры числа, можно представить следующим образом:

$$A = a_n a_{n-1} ... a_1 a_0 = a_n \cdot 10^n + a_{n-1} \cdot 10^{n-1} + ... + a_1 \cdot 10^1 + a_0 \cdot 10^0 =$$
$$= (((...((a_n \cdot 10 + a_{n-1}) \cdot 10 + a_{n-2}) \cdot 10...) \cdot 10 + a_1) \cdot 10 + a_0.$$

Например, число 1234 можно представить:

$$1234 = 1 \cdot 10^3 + 2 \cdot 10^2 + 3 \cdot 10^1 + 4 \cdot 10^0 = ((1 \cdot 10 + 2) \cdot 10 + 3) \cdot 10 + 4.$$

Из данного представления видно, что получить последнюю цифру можно, если найти остаток от деления числа на 10. В связи с этим, для разложения числа на составляющие его цифры можно использовать следующий алгоритм:

1. Находим остаток при делении числа A на 10, т.е. получаем крайнюю правую цифру числа.
2. Находим целую часть числа при делении A на 10, т.е. отбрасываем от числа A крайнюю правую цифру.
3. Если преобразованное A > 0, то переходим на пункт 1. Иначе число равно нулю, и отделять от него больше нечего.

Данный алгоритм будет использоваться при разработке нерекурсивной функции.

С другой стороны, сумму цифр числа 1234 можно представить следующим образом sum(1234)=sum(123)+4=(sum(12)+3)+4=(((sum(1)+2)+3)+4)=(((sum(0)+1)+2)+3)+4. При этом, если A=0, то сумма цифр числа также равна нулю, т.е. sum(0)=0. В противном случае, сумму цифр числа A можно представить рекуррентным соотношением sum(A)=sum(A/10)+A%10. Полученное рекуррентное соотношение будем использовать при разработке рекурсивной функции.

```
#include "iostream"
using namespace std;
unsigned long  F(unsigned long  a)  //нерекурсивная функция
{
  unsigned long  sum=0;
  while (a>0)        //пока а больше нуля
  {
    sum+=a%10;  //добавляем к сумме последнюю цифру числа а
    a/=10;          //отбрасываем от числа а последнюю цифру
  }
  return sum;              //возвращаем в качестве результата сумму цифр числа а
}
unsigned long  F_Rec(unsigned long  a)   //рекурсивная функция
{
  if (a==0) return 0;       //если а=0, то возвращаем 0
  //иначе обращаемся к рекуррентному соотношению
  else return F_Rec(a/10)+a%10;
}
int main()
{
  unsigned long a;
  cout <<"a= "; cin >>a;
  cout <<"F("<<a<<")= "<< F(a)<<endl;
  cout <<"F_Rec("<<a<<")= "<< F_Rec(a)<<endl;
  return 0;
}
```

***Пример 2*:** вычислить n-ный член последовательности Фиббоначи.

Первые два члена последовательности Фиббоначи равны 1, остальные получаются по рекуррентной формуле $a_n=a_{n-1}+a_{n-2}$.

```
#include "iostream"
using namespace std;
unsigned long F(short n)        //нерекурсивная функция
```

```
{
  if (n==1 || n==2) return 1;
  else
  {
    unsigned long a, a1=1, a2=1;
    for (short i=3; i<=n; ++i)
      {
        a=a1+a2;
        a2=a1;
        a1=a;
      }
  return a;
  }
}
unsigned long F_Rec(short  n)          //рекурсивная функция
{
  if (n==1 || n==2) return 1;
  else return F_Rec(n-1)+F_Rec(n-2);
}
int main()
{
  short n;
  cout <<"n= "; cin >>n;
  cout <<"F("<<n<<")= "<< F(n)<<endl;
  cout <<"F_Rec("<<n<<")= "<< F_Rec(n)<<endl;
  return 0;
}
```

Рассмотренные выше рекурсивные функции возвращали некоторое значение, заданное рекуррентным соотношением. Однако не все функции возвращают значение. Кроме того, рассмотренные выше функции представляют собой простой вариант рекурсивной функции. В общем случае рекурсивная функция может включать в себя некоторое множество нерекурсивных операторов и один или несколько операторов рекурсивного вызова. Рассмотрим примеры «сложных» рекурсивных функций, не возвращающих значение.

Пример 3. Для заданного значения n вывести на экран n строк, в каждой из которых содержится n звездочек. Например, для n=5 на экран нужно вывести следующую таблицу:

```
*
**
***
****
*****
```

```
#include "iostream"
using namespace std;
void Stroka(short n)       //функция выводит на экран строку из n звездочек
{
  for (short i=1; i<=n; ++i) cout <<'*';
```

```
  cout<<endl;
}
void F(short n)    //нерекурсивная функция
{
      for (short i=1;i<=n;++i)   //выводит n строк по i звездочек в каждой
            Stroka(i);
}
//рекурсивная функция, где i – номер текущей строки, n – номер последней
//строк, и при этом номер строки совпадает с количеством звездочек в строке
void F_Rec(short i,short  n)
{
 if (i<=n) //если номер текущей строки не больше номера последней строки, то
  {
    Stroka(i);         //выводим i звездочек в текущей строке и
    F_Rec(i+1,n);  //переходим к формированию следующей строки
  }
}
int main()
{
  short n;
  cout <<"n= "; cin >>n;
  cout <<"_____F_____\n";
  F(n); //
  cout <<"_____F_Rec_____\n";
  F_Rec(1,n);       //1 – это номер первой строки, n – номер последней строки
  return 0;
}
```

Пример 4. Для заданного нечетного значения n (например, для n=7) вывести на экран следующую таблицу:

```
*******
 *****
  ***
   *
   *
  ***
 *****
*******
```

Данную таблицу условно можно разделить на две части. Рассмотрим отдельно верхнюю часть:

Номер строки	Содержимое экрана	i - количество пробелов в строке	Количество звездочек в строке
0	*******	0	7
1	*****	1	5
2	***	2	3
3	*	3	1

Таким образом, если нумеровать строки с нуля, то номер строки совпадает с количеством пробелов, которые нужно напечатать в начале этой строки. Так как

количество звездочек в каждой следующей строке уменьшается на 2, то количество звездочек в строке с номером i можно определить по формуле n-2i, где n – это количество звездочек в нулевой строке. Всего нужно напечатать n/2+1 строк.

Аналогичную зависимость можно выявить и для нижней части таблицы.

```
#include "iostream"
using namespace std;
void Stroka(short n, char a)     //функция выводит на экран n раз символ a
{
  for (short i=1; i<=n; ++i)
    cout <<a;
}
void F(short n)      //нерекурсивная функция
{
    //выводим верхнюю часть таблицы, в которой в каждой строке вначале
    for (short i=0;i<=n/2;++i)
    {
      Stroka(i,' ');          //печатаем пробелы
      Stroka(n-2*i,'*');      //затем звездочки
      cout<<endl;             //затем переводим курсор на новую строку
    }
    for (int i=n/2; i>=0;--i) // аналогично выводим нижнюю часть таблицы
    {
      Stroka(i,' ');
      Stroka(n-2*i,'*');
      cout<<endl;
    }
}
//рекурсивная функция, где i определяет номер текущей строки, n – количество
//звездочек в строке
void F_Rec(short i,short  n)
{
  if (n>0 )
  {
    //действия до рекурсивного вызова – позволяют вывести верхнюю часть
    //таблицы
    Stroka(i, ' ');
    Stroka(n, '*');
    cout<<endl;
    //вызываем этот же функцию, увеличивая номер строки, и уменьшая
    //количество звездочек в ней
    F_Rec(i+1,n-2);
    //действия после рекурсивного вызова – позволяют вывести нижнюю часть
    //таблицы
    Stroka(i, ' ');
    Stroka(n, '*');
    cout<<endl;
  }
}
```

```
int main()
{
  short n;
  cout <<"n= "; cin >>n;
  cout <<"_____F_____\n";
  F(n);
  cout <<"_____F_Rec_____\n";
  F_Rec(0,n);
  return 0;
}
```

Все примеры, рассмотренные ранее, относились к прямой рекурсии. Однако существует еще и косвенная рекурсия, в которой функция вызывает себя через другую вспомогательную функцию. Продемонстрируем косвенную рекурсию на примере программы, которая для заданного значения n выводит на экран следующее за ним простое число.

Данная программа содержит функцию Prim, которая возвращает true, если ее параметр является простым числом, false – в противном случае. Чтобы установить, является ли число j простым, нужно проверить делимость числа j на все простые числа, не превышающие квадратный корень из j. Перебор таких простых чисел можно организовать следующим образом. Рассмотреть первое простое число – 2, а затем, используя функцию NextPrim, возвращающее следующее за значением ее параметра простое число, получить все простые числа, не превышающие квадрата числа j. В свою очередь, функция NextPrim обращается к функции Prim для того, чтобы определить является ли заданное число простым.

Таким образом, функции Prim и NextPrim перекрестно вызывают друг друга. В этом и проявляется косвенная рекурсия.

```
#include "iostream"
using namespace std;
int NextPrim(int i); //опережающее описание
bool Prim (int j)
{
  int k=2;    //первое простое число
  //значение k «пробегает» последовательность простых чисел, начиная с 2 до
  //корня из j, при этом проверяется делится ли j на одно из таких простых
  //чисел
  while (k*k<=j && j%k!=0)
    k=NextPrim(k);        //вызов метода NextPrim
  return (j%k==0)? false: true;
}
int NextPrim(int i)
{
  int p=i+1;
  while (!Prim(p))  //вызов метода Prim
        ++p;
  return p;
}
```

```
int main()
{
  int n;
  cout<<"n= ";
  cin >>n;
  cout<<"Следующее за "<<n<<" простое число равно " << NextPrim(n)<<endl;
  return 0;
}
```

Замечание. Обратите внимание на то, что при объявлении функции NextPrim использовалось «опережающее описание», смысл которого заключается в том, что заголовок функции указывается раннее, чем полностью описывается сама функция. Такое описание позволяет обращаться к функции NextPrim из функции Prim.

Рекурсия является удобным средством решения многих задач: сортировки числовых массивов, обхода таких структур данных как деревья и графы.

С другой стороны, применение рекурсивных функций в ряде случаев оказывается нерациональным. Вспомним рекурсивную функцию подсчета n-ного члена последовательности Фиббоначи. Данная функция будет работать весьма неэффективно. F_Rec(17) вычисляется в ней как F_Rec(16)+F_Rec(15). В свою очередь, F_Rec(16) вычисляется в ней как F_Rec(15)+F_Rec(14). Таким образом, F_Rec(15) будет вычисляться 2 раза, F_Rec(14) – 3 раза, F_Rec(13) – 5 раз и т.д. Всего для вычисления F_Rec(17) потребуется выполнить более тысячи операций сложения. Для сравнения при вычислении F(17), т.е. используя нерекурсивную функцию подсчета n-ного члена последовательности Фиббоначи, потребуется всего лишь 15 операций сложения.

Таким образом, при разработке рекурсивных функций следует задуматься об их эффективности.

2.2. Перегрузка функций

Иногда бывает удобно, чтобы функции, реализующие один и тот же алгоритм для различных типов данных, имели одно и то же имя. Использование нескольких функций с одним и тем же именем, но различными типами и количеством параметров называется *перегрузкой.* Компилятор определяет, какую именно функцию требуется вызвать, по типу и количеству фактических параметров.

Рассмотрим следующий пример:

```
#include "iostream"
using namespace std;
int max(int a)       //первая версия функции max
{
  int b = 0;
  while (a > 0)
  {
    if (a % 10 > b) b = a % 10;
    a /= 10;
  }
  return b;
}
```

```
int max(int a, int b) //вторая версия функции max
{
  if (a > b) return a;
  else return b;
}
int max(int a, int b, int c) //третья версия функции max
{
  if (a > b && a > c) return a;
  else if (b > c) return b;
       else return c;
}
int main()
{
  int a = 1283, b = 45, c = 35740;
  cout<< max(a)<<endl;
  cout<< max(a, b)<<endl;
  cout<< max(a, b, c)<<endl;
}
```

При вызове функции max компилятор выбирает вариант, соответствующий типу и количеству передаваемых в функцию параметров. Если точного соответствия не найдено, выполняются неявные преобразования типов в соответствии с общими правилами. Если преобразование невозможно, выдается сообщение об ошибке. Если выбор перегруженной функции возможен более чем одним способом, то выбирается «лучший» из вариантов (вариант, содержащий меньшие количество и длину преобразований в соответствии с правилами преобразования типов). Если существует несколько вариантов, из которых невозможно выбрать лучший, выдается сообщение об ошибке.

Замечание. Перегрузка функций является проявлением *полиморфизма*, одного из основных свойств объектно-ориентированного программирования (ОПП). Программисту гораздо удобнее помнить одно имя функции и использовать его для работы с различными типами данных, а решение о том, какой вариант функции вызвать, возложить на компилятор. Этот принцип широко используется в стандартных библиотеках C++. Например, функция *pow* заголовочного файла "*math*" имеет 7 перегруженных версий: *pow(double x, double y), pow(double x, int y), pow(float x, float y)* и т.д.

3.3. Функции-шаблоны

Многие алгоритмы не зависят от типов данных, с которыми они работают. Например, функция, вычисляющая максимальное значение в одномерном массиве, должна «уметь» обрабатывать как целочисленные, так и вещественные массивы. Для решения этой проблемы в C++ можно применить перегрузку функций. Однако в этом случае нам придется написать 9 функций – 6 для целых типов и 3 для вещественных, что не только увеличит объем программы, но и снизит ее быстродействие, т.к. компилятор будет тратить время на выбор нужной версии функции. В C++ существует другая возможность решения данной проблемы – через использование функций-шаблонов.

С помощью функций-шаблонов можно реализовать алгоритм, который будет применяться к различным типам данных, а конкретный тип данных будет

передаваться в функцию в качестве параметра на этапе компиляции. Компилятор автоматически генерирует код, соответствующий переданному типу.

Формат функции-шаблона:

```
template < typename имя_типа>
заголовок_функции
{
  тело_функции
}
```

где:

template – используется для обозначения функции-шаблона;

< typename имя_типа> - определяет параметр, который будет хранить информацию о том, какой именно тип данных был передан в шаблон;

заголовок_функции определяется стандартным образом, т.е указывается тип возвращаемого значения, имя функции и, в круглых скобках, список параметров; при этом тип возвращаемого значения и/или хотя бы один из параметров функции должен имеет тип, определенный параметром *имя_типа*.

Рассмотрим использование функций-шаблонов на следующем примере:

```
#include "iostream"
using namespace std;
//фунция-шаблон для вывода одномерного массива на экран
template < typename X>
void printArray(char *name, X *a, int n)
{
  cout<<name;
  for (int i=0; i<n; ++i)
    cout<<a[i]<<"\t";
  cout<<endl;
}
//фунция-шаблон для поиска максимального значения в одномерном массиве
 template < typename X>
X maxArray(X *a, int n)
{
  X max=a[0];
  for (int i=0; i<n; ++i)
    if (max<a[i]) max=a[i];
  return max;
}
int main()
{
  int a[]={1, -2, 4, -5, 3};
  //обращение к функциям-шаблонам с целочисленным массивом
  int maxA=maxArray( a, 5);
  printArray("Array a: ",a,5);
  cout <<"max(a)= "<<maxA<<endl;
  //обращение к функциям-шаблонам с вещественным массивом
  double b[]={3.4, -3.1, -6.5, 2};
  double maB=maxArray(b, 4);
```

```
  printArray("Array b: ",b,4);
  cout <<"max(b)= "<<maB<<endl;
}
```

В функции-шаблоне printArray параметр X используется для хранения информации о типе массива, который передается в данную функцию в качестве второго параметра.

В функции-шаблоне maxArray параметр X используется для хранения информации о типе массива, который передается в данную функцию в качестве первого параметра, а также для определения типа возвращаемого значения данной функции. При этом параметр X используется для описания переменной max в теле функции maxArray.

В общем случае функция-шаблон может содержать несколько параметров, отвечающих за информацию о типах данных. Например:

```
template < typename One, typename Two>
void F(One a, Two b)
{
  ...
}
```

Однако, для того чтобы более полно использовать механизм шаблонов, нужно изучить основы ООП.

3.4. Упражнения

I.Разработка нерекурсивных функции

Разработать функцию, которая для заданного натурального числа N возвращает значение true – если число простое, false – если число составное. С помощью данной функции:

1. вывести на экран все простые числа на отрезке [a, b];
2. найти количество всех простых чисел на отрезке [a, b];
3. найти сумму всех составных чисел на отрезке [a, b]
4. для заданного числа A вывести на экран предшествующее по отношению к нему простое число.

Разработать функцию, которая для заданного натурального числа N возвращает количество его делителей. С помощью данной функции:

5. для каждого целого числа на отрезке [a, b] вывести на экран количество делителей;
6. вывести на экран только те целые числа отрезка [a, b], у которых количество делителей равно заданному числу;
7. вывести на экран только те целые числа отрезка [a, b], у которых количество делителей максимально;
8. для заданного числа A вывести на экран следующее по отношению к нему число, имеющее столько же делителей, сколько и число A.

Разработать функцию, которая для заданного натурального числа N возвращает сумму его делителей. С помощью данной функции:

9. для каждого целого числа на отрезке [a, b] вывести на экран сумму его делителей;
10. вывести на экран только те целые числа отрезка [a, b], у которых сумма делителей равна заданному числу;
11. вывести на экран только те целые числа отрезка [a, b], у которых сумма делителей максимальна,
12. для заданного числа A вывести на экран предшествующее по отношению к нему число, сумма делителей которого равна сумме делителей числа A.

Разработать функцию, которая для заданного натурального числа N возвращает сумму его цифр. С помощью данной функции:

13. для каждого целого числа на отрезке [a, b] вывести на экран сумму его цифр;
14. вывести на экран только те целые числа отрезка [a, b], у которых сумма цифр числа равна заданному значению;
15. вывести на экран только те целые числа отрезка [a, b], у которых сумма цифр нечетная,
16. для заданного числа A вывести на экран предшествующее по отношению к нему число, сумма цифр которого равна сумме цифр числа A.

Разработать функцию, которая для заданных натуральных чисел N и M возвращает их наибольший общий делитель. С помощью данной функции:

17. сократить дробь вида a/b;
18. найти наименьшее общее кратное для двух натуральных чисел;
19. вычислить значение выражения $\frac{a}{b}+\frac{d}{c}$; результат представить в виде обыкновенной дроби, выполнив сокращение;
20. найти наибольший общий делитель для n натуральных чисел.

II. Разработать рекурсивную функцию, возвращающую значение:

1. для вычисления n-го члена следующей последовательности $b_1 = -10,\ \ b_2 = 2,\ \ b_{n+2} = |b_n| - 6b_{n+1}$.
2. для вычисления n-го члена следующей последовательности $b_1 = 5,\ \ b_{n+1} = \frac{b_n}{n^2+n+1}$.
3. для вычисления количества цифр в заданном натуральном числе.
4. для нахождения наибольшего общего делителя методом Евклида:

$$НОД(a,b) = \begin{cases} a, если\ a = b; \\ НОД(a-b,b), если\ a > b; \\ НОД(a,b-a), если\ b > a. \end{cases}$$

5. для вычисления значения функции Аккермана для неотрицательных чисел n и m. Функция Аккермана определяется следующим образом:

$$A(n,m) = \begin{cases} m+1, \text{ если } n = 0; \\ A(n-1,1), \text{ если } n \neq 0, m = 0; \\ A(n-1, A(n,m-1)), \text{ если } n > 0, m > 0. \end{cases}$$

6. для вычисления числа сочетаний $C(n, m)$, где $0 \le m \le n$, используя следующие свойства $C_n^0 = C_n^n = 1;\ C_n^m = C_{n-1}^m + C_{n-1}^{m-1}$ при $0<m<n$.

7. вычисляющую число a, для которого выполняется неравенство $2^{a-1} \le n \le 2^a$, где n – натуральное число. Для подсчета числа a использовать формулу: $a(n) = \begin{cases} 1, n = 1; \\ a(n/2) + 1, n > 1. \end{cases}$
8. для вычисления x^n (x–вещественное, $x \neq 0$, а n–целое) по формуле: $x^n = \begin{cases} 1 & при\ n = 0, \\ 1/x^{|n|} & при\ n < 0, \\ x \cdot x^{n-1} & при\ n > 0. \end{cases}$. Вычислить значение x^n для различных x и n.
9. для вычисления $\sum_{i=1}^{n} i$, где n – натуральное число. Для заданных натуральных чисел m и k вычислить с помощью разработанного метода значение выражения $\sum_{i=1}^{m} i + \sum_{i=1}^{2k} i$.
10. для вычисления значения функции $F(N) = \frac{N}{\sqrt{1+\sqrt{2+\sqrt{3+\ldots\sqrt{N}}}}}$.

    Найти ее значение при заданном натуральном N.
11. для вычисления цепной дроби: $\cfrac{x}{1+\cfrac{x}{2+\cfrac{x}{3+\ldots\cfrac{x}{n+x}}}}$. Найти значение данной дроби при заданном натуральном n.
12. для вычисления суммы цифр в строке; с помощью данной функции определить, в каком из двух предложений сумма цифр больше.
13. для вычисления количества цифр в строке; с помощью данной функции определить, в каком из двух предложений цифр больше.
14. определяющую, является ли заданная строка палиндромом.
15. определяющую, является ли палиндромом часть строки s, начиная с i-го символа и заканчивая j-м символом.
16. для перевода числа из десятичной системы счисления в двоичную;
17. для перевода числа из двоичной системы счисления в десятичную;
18. для вычисления наибольшего значения в одномерном массиве;
19. для вычисления наименьшего значения в двумерном массиве;
20. для вычисления суммы элементов в одномерном массиве.

III. Разработать рекурсивную функцию, не возвращающую значений:

1. Даны первый член и разность арифметической прогрессии. Написать рекурсивную функцию для нахождения n-го члена и суммы n первых членов прогрессии.
2. Даны первый член и знаменатель геометрической прогрессии. Написать рекурсивную функцию для нахождения n-го члена и суммы n первых членов прогрессии.

3. Разработать рекурсивную функцию, которая по заданному натуральному числу N выведет на экран все натуральные числа, не превышающие N, в порядке возрастания. Например, для N=8, на экран выводится 1 2 3 4 5 6 7 8.
4. Разработать рекурсивную функцию, которая по заданному натуральному числу N выведет на экран все натуральные числа, не превышающие N, в порядке убывания. Например, для N=8, на экран выводится 8 7 6 5 4 3 2 1.
5. Разработать рекурсивную функцию для вывода на экран стихотворения:

10 лунатиков жили на луне
10 лунатиков ворочались во сне
Один из лунатиков упал с луны во сне
9 лунатиков осталось на луне
9 лунатиков жили на луне
9 лунатиков ворочались во сне
Один из лунатиков упал с луны во сне
8 лунатиков осталось на луне
.....
И больше лунатиков не стало на луне

6. Дано натуральное число n. Разработать рекурсивную функцию для вывода на экран следующей последовательности чисел:

```
1
2  2
3  3  3
...
n  n  n  ...  n
```

7. Дано натуральное число n. Разработать рекурсивную функцию для вывода на экран следующей последовательности чисел:

```
1
2     1
3     2     1
...
n     n-1   n-2   ...   1
```

8. Разработать рекурсивную функцию для вывода на экран цифр натурального числа в прямом порядке. Применить эту функцию ко всем числам из интервала от A до B.
9. Разработать рекурсивную функцию для вывода на экран всех делителей заданного натурального числа n.
10. Дано натуральное четное число n. Разработать рекурсивную функцию для вывода на экран следующей картинки:

```
*********      (n звездочек)
********       (n-1 звездочка)
*******        (n-2 звездочки)
...
*              (1 звездочка)
```

11. Дано натуральное четное число n. Разработать рекурсивную функцию для вывода на экран следующей картинки:

*********	(0 пробелов, n звездочек)
********	(1 пробел, n-1 звездочка)
*******	(2 пробела, n-2 звездочки)
...	
*	(n-1 пробел, 1 звездочка)

12. Дано натуральное четное число n. Разработать рекурсивную функцию для вывода на экран следующей картинки:

* *	(n пробелов между звездочками)
** **	(n-2 пробела)
*** ***	(n-4 пробела)
...	...
***** *****	(2 пробела)
**********	(0 пробелов)
***** *****	(2 пробела)
...	...
*** ***	(n-4 пробела)
** **	(n-2 пробела)
* *	(n пробелов

13. Дано натуральное число n. Разработать рекурсивную функцию для вывода на экран следующей картинки:

1	(1 раз)
222	(3 раза)
33333	(5 раз)
...	(n раз)
33333	(5 раз)
222	(3 раза)
1	(1 раз)

14. Разработать рекурсивную функцию для вывода на экран следующей картинки:

AAAAAAAAAA...AAAAAAAAAA	(80 раз)
BBBBBBBBB...BBBBBBBB	(78 раз)
CCCCCCCC...CCCCCCCC	(76 раз)
...	...
YYY...YYY	(32 раза)
ZZ...ZZ	(30 раз)
YYY...YYY	(32 раза)
...	...
CCCCCCCC...CCCCCCCC	(76 раз)
BBBBBBBBB...BBBBBBBB	(78 раз)
AAAAAAAAAA...AAAAAAAAAA	(80 раз)

15. Разработать рекурсивную функцию, которая удаляет из заданной строки все точки.
16. Разработать рекурсивную функцию, которая после каждого вхождения символа a в строку s добавляет символ b.
17. Разработать рекурсивную функцию, которая в заданной строке заменяет все слова, начинающиеся с заглавной буквы, на многоточие.
18. Разработать рекурсивную функцию, которая каждый отрицательный элемент одномерного массива заменяет противоположным по значению элементом.
19. Разработать рекурсивную функцию, которая каждый четный элемент двумерного массива заменяет нулем.

20. Разработать рекурсивную функцию для нахождения максимального элемента и его номера в одномерном массиве.

IV. Используя механизм перегрузки функций, разработайте две версии функции F, заголовки которых выглядят следующим образом:

1) float F(float x); 2) void F (float x, float &y);

Продемонстрируйте работу данных функций на примерах.

1. $y=\begin{cases}\dfrac{1}{(0.1+x)^2}, & \text{если } x\ge 0.9;\\ 0.2x+0.1, & \text{если } 0\le x<0.9;\\ x^2+0.2, & \text{если } x<0.\end{cases}$

2. $y=\begin{cases}\sin(x), & \text{если } |x|<3;\\ \dfrac{\sqrt{x^2+1}}{\sqrt{x^2+5}}, & \text{если } 3\le|x|<9;\\ \sqrt{x^2+1}-\sqrt{x^2+5}, & \text{если } |x|\ge 9.\end{cases}$

3. $y=\begin{cases}0, & \text{если } x<a;\\ \dfrac{x-a}{x+a}, & \text{если } x>a;\\ 1, & \text{если } x=a.\end{cases}$

4. $y=\begin{cases}x^3-0.1, & \text{если } |x|\le 0.1;\\ 0.2x-0.1, & \text{если } 0.1<|x|\le 0.2;\\ x^3+0.1, & \text{если } |x|>0.2.\end{cases}$

5. $y=\begin{cases}a+b, & \text{если } x^2-5x<0;\\ a-b, & \text{если } 0\le(x^2-5x)<10;\\ ab, & \text{если } x^2-5x\ge 10.\end{cases}$

6. $y=\begin{cases}x^2, & \text{если } (x^2+2x+1)<2;\\ \dfrac{1}{x^2-1}, & \text{если } 2\le(x^2+2x+1)<3;\\ 0, & \text{если } (x^2+2x+1)\ge 3.\end{cases}$

7. $y=\begin{cases}-4, & \text{если } x<0;\\ x^2+3x+4, & \text{если } 0\le x<1;\\ 2, & \text{если } x\ge 1.\end{cases}$

8. $y=\begin{cases}x^2-1, & \text{если } |x|\le 1;\\ 2x-1, & \text{если } 1<|x|\le 2;\\ x^5-1, & \text{если } |x|>2.\end{cases}$

9. $y=\begin{cases}(x^2-1)^2, & \text{если } x<1;\\ \dfrac{1}{(1+x)^2}, & \text{если } x>1;\\ 0, & \text{если } x=1.\end{cases}$

10. $y=\begin{cases}x^2, & \text{если } (x+2)\le 1;\\ \dfrac{1}{x+2}, & \text{если } 1<(x+2)<10;\\ x+2, & \text{если } (x+2)\ge 10;\end{cases}$

11. $y=\begin{cases}x^2+5, & \text{если } x\le 5;\\ 0, & \text{если } 5<x<20;\\ 1, & \text{если } x\ge 20.\end{cases}$

12. $y=\begin{cases}0, & \text{если } x<0;\\ x^2+1, & \text{если } x\ge 0 \text{ и } x\ne 1;\\ 1, & \text{если } x=1.\end{cases}$

13. $y=\begin{cases}1, & \text{если } x=1 \text{ или } x=-1;\\ \dfrac{-1}{1-x}, & \text{если } x\ge 0 \text{ и } x\ne 1;\\ \dfrac{1}{1+x}, & \text{если } x<0 \text{ и } x\ne -1.\end{cases}$

14. $y=\begin{cases}0.2x^2-x-0.1, & \text{если } x<0;\\ \dfrac{x^2}{x-0.1}, & \text{если } x>0 \text{ и } x\ne 0.1;\\ 0, & \text{если } x=0.1.\end{cases}$

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

16. $y=\begin{cases}x, & \text{если } x>0;\\ 0, & \text{если } -1\le x\le 0;\\ x^2, & \text{если } x<-1.\end{cases}$

17. $y=\begin{cases}a+bx, & \text{если } x<93;\\ b-ac, & \text{если } 93\le x\le 120;\\ abx, & \text{если } x>120.\end{cases}$

18. $y=\begin{cases}x^2-0.3, & \text{если } y<3;\\ 0, & \text{если } 3\le x\le 5;\\ x^2+1, & \text{если } x>5.\end{cases}$

19. $y=\begin{cases}\sqrt{5x^2+5}, & \text{если } |x|<2;\\ \dfrac{|x|}{\sqrt{5x^2+5}}, & \text{если } 2\le|x|<10;\\ 0, & \text{если } |x|\ge 10.\end{cases}$

20. $y=\begin{cases}\sin(x), & \text{если } |x|<\dfrac{\pi}{2};\\ \cos(x), & \text{если } \dfrac{\pi}{2}\le|x|\le\pi;\\ 0, & \text{если } |x|>\pi.\end{cases}$

V. Использование функций-шаблонов: для работы с двумерными массивами арифметических типов данных разработать шаблоны ввода и вывода массива, а также шаблон для решения основной задачи:

1. Заменить все положительные элементы противоположными им числами.
2. Заменить все элементы, меньшие заданного числа, этим числом.
3. Заменить все элементы, попадающие в интервал [a, b], нулем.
4. Все элементы, меньшие заданного числа, увеличить в два раза.
5. Подсчитать среднее арифметическое элементов.
6. Подсчитать среднее арифметическое отрицательных элементов.
7. Подсчитать сумму элементов, попадающих в заданный интервал.
8. Подсчитать количество элементов, не попадающих в заданный интервал.
9. Подсчитать количество максимальных элементов.
10. Заменить все минимальные элементы противоположными по значению.
11. Подсчитать среднее арифметическое элементов, расположенных выше главной диагонали.
12. Подсчитать сумму элементов, расположенных на побочной диагонали.
13. Подсчитать среднее арифметическое ненулевых элементов, расположенных над побочной диагональю.
14. Подсчитать среднее арифметическое элементов, расположенных под побочной диагональю.
15. Поменять местами столбцы по правилу: первый с последним, второй с предпоследним и т.д.
16. Если количество строк в массиве четное, то поменять строки местами по правилу: первую строку со второй, третью – с четвертой и т.д. Если количество строк в массиве нечетное, то оставить массив без изменений.
17. Подсчитать норму матрицы по формуле $\|A\| = \sum_i \max_j a_{i,j}$.
18. Подсчитать норму матрицы по формуле $\|A\| = \sum_j \max_i a_{i,j}$.

19.Вывести элементы матрицы в следующем порядке: 

20.Выяснить, является ли матрица симметричной относительно главной диагонали.

Замечание. Продемонстрировать использование шаблонов на нескольких примерах.

3.5. Самостоятельная работа

Задача 1. Разработать рекурсивную функцию для вывода на экран всех возможных разложений натурального числа n на множители (без повторений). Например, для n=12 на экран может быть выведено:

2*2*3=12
2*6=12
3*4=12

Задача 2. Разработать рекурсивную функцию для вывода на экран всех возможных разложений натурального числа n на слагаемые (без повторений). Например, для n=5 на экран может быть выведено:

1+1+1+1+1=5
1+1+1+2=5
1+1+3=5
1+4=5
2+1+2=5
2+3=5

Задача 3. Разработать рекурсивную функцию для вычисления определителя заданной матрицы, пользуясь формулой разложения по первой строке: $\det(A)=\sum_{k=1}^{n}(-1)^{k+1}a_{1k}\det(B_k)$. Продемонстрируйте работу данной функции на примерах.

4. ОРГАНИЗАЦИЯ ФАЙЛОВОГО ВВОДА/ВЫВОДА

Мы уже знаем, что в языке C++ механизм ввода-вывода функционирует с помощью потоков. Поток – это абстрактное понятие, относящееся к любому переносу данных от источника к приемнику. Чтение данных из потока называется извлечением, вывод в поток – помещением или включением. По направлению обмена потоки можно разделить на входные (данные помещаются в поток), выходные (извлекаются из потока) и двунаправленные потоки (допускается как извлечение, так и включение).

По виду устройств, с которым работает поток, потоки можно разделить на стандартные, файловые и строковые.

Стандартные потоки предназначены для передачи данных от клавиатуры и на экран дисплея. До сих пор ввод-вывод мы осуществляли с помощью стандартных потоков.

Файловые потоки предназначены для обмена информацией с файлами на внешних носителях. Организацию файловых потоков мы рассмотрим в данном разделе.

Строковые потоки позволяют считывать и записывать информацию из областей оперативной памяти.

Замечание. Изучение строковых потоков выходит за рамки данного пособия.

В общем случае, потоки C++ обеспечивают надежную работу как со стандартными, так и с определенными пользователями типами данных, а также единообразный и понятный синтаксис.

Для поддержки потоков библиотека C++ содержит иерархию классов, построенную на основе двух базовых классов – ios и streambuf. Класс ios содержит общие для ввода и вывода поля и функции. Класс streambuf обеспечивает взаимодействие потоков с физическими устройствами.

Замечание. Поток определяется как последовательность байтов и не зависит от конкретного устройства, с которым производится обмен данных (оперативной памяти, файла на диске, клавиатуры, принтера или экрана). Обмен с потоком для увеличения скорости передачи данных производится через специальную область оперативной памяти – буфер. Фактическая передача данных выполняется при выводе после заполнения буфера, при вводе – если буфер исчерпан.

От этих классов наследуется класс istream для входных потоков и ostream – для выходных потоков. Два последних класса являются базовыми для класса iostream, реализующего двунаправленные потоки. Напомним, что именно класс iostream мы использовали для организации стандартного ввода/вывода данных.

В C++ реализованы три класса для работы с файлами: ifstream – класс входных файловых потоков, ofstream – класс выходных файловых потоков; fstream – класс двунаправленных файловых потоков. Эти классы являются производными от классов istream, ostream и iostream соответственно, поэтому они наследуют перегруженные операции << и >>, а также манипуляторы для управления форматированием и размещением данных.

Рассмотрим классы для работы с файлами более подробно.

4.1. Файловые потоки

Файл – это поименованная совокупность данных на внешнем носителе информации, например, на жестком диске. Логически файл можно представить как конечное количество последовательных байтов.

Физически файл начинается с некоторого байта памяти. При открытии файла его внутренний указатель устанавливается на начальный байт файла, который имеет индекс ноль. Каждый следующий байт файла имеет индекс на единицу больше. Чтение/запись данных производится в текущую позицию указателя. Индекс последнего байта файла определяет его размер в байтах.

По способу доступа файлы можно разделить на последовательные, чтение и запись в которых производится с начала файла байт за байтом, и файлы с произвольным доступом, допускающие чтение и запись в указанную позицию.

По внутреннему представлению данных файлы бывают текстовые и двоичные.

В текстовых файлах все данные сохраняются в виде символов, даже числа. При считывании данных из потока (или записи данных в поток) происходят необходимые преобразования типов. Например, при чтении данных строки могут преобразовываться к арифметическим типам, а при записи, наоборот, арифметические типы преобразуются в строковый тип, на что потребуются

дополнительные временные затраты. Однако, текстовые файлы просты для чтения пользователем, а для их создания и редактирования можно пользоваться текстовым редактором, например, «Блокнот».

В двоичных файлах данные сохраняются во внутреннем (машинном) представлении, поэтому они обычно занимают меньше места, и работа с ним происходит быстрее, т.к. преобразование типов не производится. Однако просмотр и редактирование двоичных файлов возможно только программным путем.

Работу с файлами через потоки можно осуществить, подключив к программе заголовочный файл "fstream". В общем случае, при программном использовании файлов предполагаются следующие последовательные этапы: создание потока нужного типа; открытие потока и связывание с ним файла; обмен данных (ввод-вывод); закрытие файла. Рассмотрим данные этапы более подробно.

Работа с текстовыми файлами

Напомним, что существуют однонаправленные потоки (потоки ввода и потоки вывода) и двунаправленные потоки. Чтобы открыть входной поток, необходимо объявить потоковый объект типа ifstream. Для открытия выходного потока нужно объявить потоковый объект типа ofstream. Поток, который предполагается использовать для операций как ввода, так и вывода, должен быть объявлен как fstream. Например, при выполнении следующего фрагмента кода будет создан входной поток, выходной поток и поток, позволяющий выполнять операции в обоих направлениях:

```
ifstream in;        // входной поток in
ofstream out;       // выходной поток out
fstream both;       // поток ввода-вывода both
```

Создав поток, его нужно связать с файлом. Это можно сделать с помощью функции *open*. Например:

```
in.open("infile.txt");
```

открывает файл по имени infile.txt, расположенный в текущем каталоге, и связывает его с входным потоком in.

Можно одновременно выполнить две операции - открыть файл и связать его с потоком. Например:

```
// открываем входной поток in и связываем его с файлом infile.txt.
ifstream in ("infile.txt");

//открываем выходной поток out и связываем его с файлом outfile.txt
ofstream out ("outfile.txt");

//открываем двунаправленный поток both и связываем с файлом bothfile.txt
fstream both ("bothfile.txt");
```

Открыв файл, можно проверить, не произошло ли при этом ошибки. Например:

```
if (!in)            //если объект in не готов к применению
{
  cout << error;    //вывести сообщение об ошибке
  return -1;        //возвратить значение -1, которое сигнализирует об ошибке
}
```

Считывать данные из файла можно с помощью операции >>, а записывать – с помощью операции <<. Например:

```
in >> x;     // считываем значение из потока in в переменную x
out << x;    // помещаем значение переменной x в поток out
```

Заметим, что при использовании операции >> для считывания данных из текстовых файлов, извлечение из потока прекращается при встрече пробельного символа (пробел, табуляция, конец строки), причем сами пробельные символы пропускаются. Чтобы избежать этого, необходимо работать с файлами в двоичном режиме доступа.

Очень часто при считывании данных из файла их количество неизвестно, но требуется считать и обработать все данные. Например, нам необходимо подсчитать среднее арифметическое всех элементов файла, в котором записаны вещественные числа. Для этого нужно считывать числа по одному до тех пор, пока в файле не останется ни одного непрочитанного числа. Предположим, что с файлом, содержащим набор чисел, соединен поток in. Тогда алгоритм вычисления среднего арифметического всех чисел из файла может быть таким:

```
double next, sum = 0;
int count = 0:
while (in >> next)  //1
{
sum = sum + next;
count++:
}
double sr = sum/count;
```

В этом цикле выражение in >>next (строка 1) служит одновременно и для считывания очередного числа из потока in, и для проверки условия окончания цикла while. Таким образом, данное выражение является и оператором, выполняющим некоторое действие, и логическим выражением. Как оператор, оно считывает число из входного потока, а как логическое выражение - возвращает значение true или false. Если в потоке имеется еще одно число, оно считывается, и логическое выражение оказывается истинным, в результате чего тело цикла выполняется еще один раз. Если же чисел больше не осталось, ничего не вводится, и логическое выражение оказывается ложным, в результате чего цикл завершается. В этом примере переменная next имеет тип double, но такой метод проверки на конец файла одинаково действует и для других типов данных, таких как int или char.

Для завершения работы с потоком его необходимо закрыть. Это можно сделать с помощью функции close(). Например:

```
in.close();   //закрывает входной поток, связанный с файлом infile.txt
out.close(); //закрывает выходной поток, связанный с файлом outfile.txt
```

Работа с двоичными файлами

Для открытия двоичных файлов необходимо открыть входной поток, используя спецификатор режима *ios :: binary*. Например:

```
ifstream in ("infile.dat", ios::binary);
```

Замечание. Функции обработки двоичных файлов могут применяться и для работы с файлами, открытыми в текстовом режиме доступа, но при этом могут иметь место преобразования символов, которые сводят на нет основную цель выполнения двоичных файловых операций.

На нижнем уровне двоичного ввода/вывода находятся функции get() и put().

Функция get () считывает один символ (один байт) из соответствующего потока и помещает его значение в переменную *ch* типа char, при этом возвращает ссылку на поток, связанный с предварительно открытым файлом. При считывании символа конца файла данная функция возвратит вызывающему потоку значение false. Обращение к функции выглядит следующим образом:

```
in.get(ch);
```

Функция put () записывает символ *ch* в поток и возвращает ссылку на этот поток. Обращение к функции выглядит следующим образом:

```
in.put(ch);
```

Рассмотрим следующий программный фрагмент:

```
char next;
in.get(next):
```

Если считываемым символом является пробел, приведенный код не пропускает его, а прочитывает и присваивает переменной next значение, равное символу пробела. Если следующим символом является символ перевода строки '\n', который означает, что программа достигла конца вводимой строки, вызов in.get(next) присваивает переменной next значение '\n'.

Замечание. Напомним, что хотя символ перевода строки записывается как пара символов '\n', в C++ они интерпретируются как один управляющий символ.

Мы уже сталкивались с вопросом, как считать все данные до конца файла. Для двоичных файлов можно использовать функцию eof.

Замечание. Название функции eof представляет собой сокращение от англ. end of file — конец файла.

В конце каждого файла имеется специальный маркер конца файла. У функции eof нет аргументов, и она возвращает значение true - если прочитан маркер конца файла, false – в противном случае. Обратиться к функции можно следующим образом – in.eof().

Рассмотрим в качестве примера следующий оператор:

```
if (!in.eof())  cout << "Not end.":
 else cout << "End of the file.";
```

Логическое выражение после ключевого слова if означает «не достигнут конец файла, связанного с потоком in». Поэтому, если программа еще не достигла конца файла, связанного с потоком in, то приведенный оператор выведет на экран следующее: Not end. Если же программа достигла конца файла, этот оператор выведет: End of the file.

Тогда содержимое файла может быть выведено на экран с помощью следующего цикла while:

```
in.get(next);
while (! in.eof())
{
  cout << next;
  in.get(next);
}
```

Этот цикл while считывает в переменную next типа char побайтно данные из файла и выводит их на экран. Когда достигается конец файла, значение выражения in.eof() становится равным true. Поэтому выражение (!in.eof()) принимает значение false, и цикл завершается. Предположим, что файл содержит следующий текст:

```
ab
c
```

На самом деле этот файл содержит четыре символа:

```
ab'\n'c
```

Приведенный выше цикл прочитает из файлового потока и выведет на экран символ "a", затем — символ "b" , далее прочитает и выведет на экран символ перевода строки "\n", а потом — символ "c". К этому моменту цикл прочитает все имеющиеся в файле символы, но выражение in.eof() по-прежнему будет возвращать false. На следующем шаге программа прочитает маркер конца файла, и цикл завершит свою работу. Вот почему в нашем цикле функция in.get (next) стоит в конце.

Рассмотренный фрагмент программы можно упростить, используя функцию *peek()*. Функция peek() возвращает следующий символ потока, или значение EOF, если достигнут конец файла.

Замечание. Напомним, что в C++ прописные и строчные буквы в имени идентификатора различаются. Поэтому eof и EOF это два совершенно разных идентификатора: eof –это имя функции, которая определяет, достигнут ли конец файла, а EOF – это константа, в которой хранится значение маркера конца файла.

Тогда содержимое файла может быть выведено на экран с помощью следующего цикла

```
while (in.peek()!=EOF)
{
  in.get(next);
  cout << next;
}
```

Функции get и put используются для считывания данных побайтно, что позволяет нам обрабатывать текстовые данные, представленные в ASCII кодировке (один символ – один байт). Однако текстовые данные могут представляться и в кодировке Unicode (один символ – два байта). Кроме того, в двоичных файлах могут храниться последовательности целых и вещественных типов, где каждое значение в машинном представлении занимает от 2 до 10 байт, а также пользовательские типы, размеры которых заранее неизвестны. В этом случае использование функций get и put невозможно.

Для считывания и записи блоков двоичных данных используются функции read() и write(), которые также являются функциями-членами потоковых классов

соответственно для ввода и для вывода. Прототипы данных функций выглядят следующим образом:

```
istream &read(char *<буфер>, <число_байт>);
ostream &write(const char *<буфер>, <число_байт>);
```

Функция read() считывает из вызывающего потока столько байт, сколько задано параметром <число_байт> и передает их в <буфер>. Если конец файла достигнут до того, как было считано нужное количество байт, то выполнение функции read() прекращается, а в буфере оказывается столько байт, сколько их было в файле. Узнать, сколько символов было считано, можно обратившись к входному потоку с помощью функции-члена gcount(). Обратиться к функции read() для открытого входного потока in можно следующим образом:

```
double i;     //вместо double может быть указан любой другой тип данных
in.read((char*) &i, sizeof(double));
```

Функция write() записывает в соответствующий поток из <буфер> столько байт, сколько задано параметром <число_байт>. Обратиться к функции write() для открытого выходного потока out можно следующим образом:

```
int i;  //вместо int может быть указан любой другой тип данных
out.write((char* )&i, sizeof(int));
```

Рассмотрим следующую программу:

```
#include "fstream"
using namespace std;
int main()
{
 //работа с текстовым файлом
 ofstream out("outfile.txt");
 for (double i=0;i<10; i+=0.5)
  out <<i<<' ';              //запись даных в текстовый файл
 out.close();
 //работа с двоичным файлом
 out.open("outfile.dat", ios::binary);
 for (double i=0;i<10; i+=0.5)
  out.write((char* )&i, sizeof(double));       //запись данных в двоичный файл
 out.close();
}
```

Если открыть созданные файлы в текстовом редакторе, то мы увидим следующие данные:

file.txt

0 0.5 1 1.5 2 2.5 3 3.5 4 4.5 5 5.5 6 6.5 7 7.5 8 8.5 9 9.5

file.dat

```
00000000  00 00 00 00 00 00 00 00  00 00 00 00 00 00 E0 3F  ...............?
00000010  00 00 00 00 00 00 F0 3F  00 00 00 00 00 00 F8 3F  .......?.......?
00000020  00 00 00 00 00 00 00 40  00 00 00 00 00 00 04 40  .......@.......@
00000030  00 00 00 00 00 00 08 40  00 00 00 00 00 00 0C 40  .......@.......@
00000040  00 00 00 00 00 00 10 40  00 00 00 00 00 00 12 40  .......@.......@
00000050  00 00 00 00 00 00 14 40  00 00 00 00 00 00 16 40  .......@.......@
00000060  00 00 00 00 00 00 18 40  00 00 00 00 00 00 1A 40  .......@.......@
00000070  00 00 00 00 00 00 1C 40  00 00 00 00 00 00 1E 40  .......@.......@
00000080  00 00 00 00 00 00 20 40  00 00 00 00 00 00 21 40  ...... @......!@
00000090  00 00 00 00 00 00 22 40  00 00 00 00 00 00 23 40  ......"@......#@
000000a0
```

Таким образом, в текстовом файле отображаются сами числа, а в двоичном их машинное представление. Как видим, машинное представление «не читаемо» для пользователя. Прочитать данные из бинарного файла можно только программным путем. Например, следующим образом:

```
#include "iostream"
#include "fstream"
using namespace std;
int main()
{
  ifstream in("file.dat", ios::binary); //файл file.dat должен существовать
  double i;
  while (in.read((char*) &i, sizeof(double))) //чтение данных из файла
      cout<<i<<' ';
  in.close();
}
```

Замечание. Обратите внимание на то, что чтение данных до конца файла осуществляется аналогично тому, как производилось чтение в текстовом файле. Только вместо оператора in >> i, для двоичного файла в условие завершения стоит оператор чтения in.read((char*) &i, sizeof(double)).

Произвольный доступ

До сих пор мы работали с файлами в режиме последовательного доступа. Рассмотрим теперь, как организовать доступ к произвольной компоненте файла.

Замечание. Произвольный доступ имеет смысл только для бинарных файлов, в которых можно точно определить размер компонент файла в байтах. В текстовых файлах в общем случае длина каждой компоненты может быть различной.

Для обеспечения произвольного доступа используется внутренний указатель файла, который определяет текущий индекс обрабатываемого байта файла (напомним, что байты файла нумеруются как и элементы массива, с нуля), и две пары функций. Одна пара функций - seekg(), seekp() – используется потоками ввода и устанавливает указатель в нужную позицию, а вторая пара функций - tellp(), tellg() – используется потоками вывода и сообщает текущую позицию указателя.

Замечание. Фактически это две функции – seek и tell, каждая из которых используется с двумя разными суффиксами: g (getting) означает получение данных, p (putting) - помещение (запись) данных.

Функция *seekg(n)* устанавливает указатель в позицию с номером n внутри файла для чтения. Обращение к ней для открытого входного потока in может выглядеть следующим образом:

```
in.seekg(3);        //указатель будет установлен на позицию с номером 3
```

Функция *seekp(n)* устанавливает указатель на позицию с номером n внутри файла для записи. Обращение к ней для открытого выходного потока out может выглядеть следующим образом:

```
out.seekp(3);       //указатель будет установлен на позицию с номером 3
```

С помощью констант позиционирования:
ios::beg – позиционирование относительно начала потока
ios::cur – позиционирование относительно текущей позиции в потоке
ios::end – позиционирование относительно конца потока
можно уточнить позицию указателя в обрабатываемом потоке. Например:

```
//Установить указатель на позицию с номером 3 относительно начала файла
in.seekg(3, ios::beg);
//Установить указатель на позицию с номером 3 относительно конца файла
in.seekg(3, ios::end);
//Перемещаем указатель на 3 индекса относительно текущей позиции
in.seekg(3, ios::cur);
```

Замечание. Константы позиционирования beg, cur перемещают указатель в направлении от начала к концу файла, а константа end – от конца к началу файла.

Функция *tellg()* возвращает текущую позицию указателя потока ввода. Обращение к ней для открытого входного потока in может выглядеть следующим образом:

```
streampos n=in.tellg();  //в переменную n запишется текущая позиция чтения
```

Функция *tellp()* возвращает текущую позицию маркера потока вывода. Обращение к ней для открытого выходного потока out может выглядеть следующим образом:

```
streampos n=in.tellp(); //в переменную n запишется текущая позиция записи
```

Замечание. streampos – специальный тип данных, который используется для хранения индекса указателя в файле. Определен в заголовочном файле "fstream".

4.2. Примеры решения задач с использованием файлового ввода/вывода

1. Дан текстовый файл f.txt. Переписать в файл g.txt все его строки в перевернутом виде.

Замечание. Напоминаем, что текстовый файл должен быть предварительно создан в текстовом редакторе, например «Блокнот».

```
#include "fstream"
#include "iostream"
#include "string"
using namespace std;
int main()
{
  ifstream in("f.txt");
  ofstream out ("g.txt");
  string s;
  while (in.peek()!=EOF)//пока не прочитан маркер конца файла
  {
```

```
    getline(in,s);    //читаем очередную стоку из файла f.txt
    for (unsigned int i=0; i<s.length()/2; i++) //зеркально отображаем строку
    {
      char a=s[i];
      s[i]=s[s.length()-1-i];
      s[s.length()-1-i]=a;
    }
    out<<s<<endl; //записываем измененную строку в файл g.txt
  }
  in.close();
  out.close();
  return 0;
}
```

f.txt

```
Многие вещи нам непонятны не потому,
что наши понятия слабы; но потому, что сии
вещи не входят в кругозор наших понятий.
Козьма Прутков,
Плоды раздумья, мысль 66
```

g.txt

```
,умотоп ен ынтянопен ман ищев еигонМ
иис отч ,умотоп он ;ыбалс яитяноп ишан отч
.йитяноп хишан розогурк в тядохв ен ищев
,воктурП амьзоК
66 ылсым ,яьмудзар ыдолП
```

2. Дан файл f.txt, компонентами которого являются целые числа. В файл g.txt переписать все неотрицательные компоненты файла f.txt, кратные трем.

```
#include "fstream"
#include "iomanip"
using namespace std;
int main()
{
  ifstream in ("f.txt");
  ofstream out("g.txt");
  int i;
  while (in.peek()!=EOF) //пока не прочитан маркер конца файла
    {
       in >>i;        //читаем из потока очередную компоненту
       if ((i>=0) && (i%3==0)) //если компонента отвечает заданным требованиям,
         out <<setw(5) << i;      //то помещаем ее в выходной поток
    }
  in.close(); out.close();
  return 0;
}
```

f.txt

```
-9 -8
-7 6 7
8
9
```

g.txt

```
  6   9
```

Замечание. При формировании выходного потока мы использовали манипулятор для форматированного вывода setw. Вспомните, что делает данный манипулятор, и подумайте, что будет, если его не использовать.

3. Дан файл f.txt, в котором записан текст. Переписать этот текст в файл f.txt с сохранением форматирования, заменив при этом все вхождения символа С на С++.

Замечание. При работе с текстовым файлом мы использовали бинарную функцию get() для считывания файла посимвольно.

```
#include "fstream"
#include "iostream"
using namespace std;
int main()
{
  char symbol;
  ifstream in("f.txt");
  ofstream out("g.txt");
  while (in.peek()!=EOF)
   {
      in.get(symbol);     //читаем очередной символ из файла
      if (symbol=='C')    //если он равен символу С
        out << "C++"; //то вместо него в выходной поток помещаем строку С++
      else out << symbol; //иначе в выходной поток помещаем текущий символ
   }
  in.close();
  out.close();
  return 0;
}
```

f.txt

Abc CBA C+c

g.txt

Abc C++BA C+++c

4. Создать бинарный файл f.dat, записав в него вещественные числа из интервала от a до b с шагом h. Вывести на экран компоненты файла f.dat через одну, начиная со второй:

Замечание. Напоминаем, что двоичные файлы создаются и просматриваются только программным путем.

```
#include "iostream"
#include "fstream"
using namespace std;
int main()
{
  ofstream out("f.dat", ios::binary);
  double a,b,h, i;
  cout <<"a= "; cin>>a;
  cout <<"b= "; cin>>b;
  cout <<"h= "; cin>>h;
  for (i=a; i<=b; i+=h) //записываем данные в двоичный файл
    out.write((char *)&i, sizeof(i));
  out.close();
  ifstream in("f.dat", ios::binary);
  //смещаем указатель относительно начала файла на столько байтов,
  //сколько отводится для хранения вещественного числа, тем самым
  // перемещаем указатель на второе вещественное число в файле
  in.seekg(sizeof(double));
```

```
  while (in.peek()!=EOF)
  {
    in.read((char*) &i, sizeof(double)); //читаем данные из двоичного файла
    cout<<i<<' ';
    //смещаем указатель относительно текущей позиции указателя на
    // столько байтов, сколько отводится для хранения вещественного числа,
    // тем самым пропускаем одно вещественное число в файле
    in.seekg(sizeof(double), ios::cur);
  }
 in.close();
 return 0;
}
```

входные данные	*выходные данные*
a=0 b=10 h=0.5	0.5 1.5 2.5 3.5 4.5 6.5 7.5 8.5 9.5

4.3. Упражнения

Замечание. Двоичные файлы создаются программным путем. Тестовые файлы нужно предварительно создать в текстовом редакторе, например, «Блокнот».

I. Работа с текстовыми файлами.

1. Дан текстовый файл. Найти количество строк, которые начинаются с данной буквы.
2. Дан текстовый файл. Найти количество строк, которые начинаются и заканчиваются одной буквой.
3. Дан текстовый файл. Найти самую длинную строку и ее длину.
4. Дан текстовый файл. Найти самую короткую строку и ее длину.
5. Дан текстовый файл. Найти номер самой длинной строки.
6. Дан текстовый файл. Найти номер самой короткой строки.
7. Дан текстовый файл. Выяснить, имеется ли в нем строка, которая начинается с данной буквы. Если да, то напечатать ее.
8. Дан текстовый файл. Напечатать первый символ каждой строки.
9. Дан текстовый файл. Напечатать символы с k1 по k2 в каждой строке.
10. Дан текстовый файл. Напечатать все нечетные строки.
11. Дан текстовый файл. Напечатать все строки, в которых имеется хотя бы один пробел.
12. Дан текстовый файл. Напечатать все строки, длина которых равна данному числу.
13. Дан текстовый файл. Напечатать все строки, длина которых меньше заданного числа.
14. Дан текстовый файл. Напечатать все строки с номерами от k1 до k2.
15. Дан текстовый файл. Получить слово, образованное k-ыми символами каждой строки.
16. Дан текстовый файл. Переписать в новый файл все его строки, вставив в конец каждой строки ее номер.

17. Дан текстовый файл. Переписать в новый файл все его строки, вставив в конец каждой строки количество символов в ней.
18. Дан текстовый файл. Переписать в новый файл все его строки, длина которых больше заданного числа.
19. Дан текстовый файл. Переписать в новый файл все его строки четной длины.
20. Дан текстовый файл. Переписать в новый файл все его строки, удалив из них символы, стоящие на четных местах.

II. Работа с текстовыми файлами.

1. Дан файл f, компонентами которого являются целые числа. Переписать все четные числа в файл g, нечетные – в файл h.
2. Дан файл f, компонентами которого являются целые числа. Переписать все отрицательные числа в файл g, положительные – в файл h.
3. Даны два файла с числами. Поменять местами их содержимое (использовать вспомогательный файл).
4. Даны два файла с числами. Получить новый файл, каждый элемент которого равен сумме соответствующих компонентов заданных файлов (количество компонентов в исходных файлах одинаковое).
5. Даны два файла с числами. Получить новый файл, каждый компонент которого равен наибольшему из соответствующих компонентов заданных файлов (количество компонентов в исходных файлах одинаковое).
6. Даны два файла с числами. Получить новый файл, каждый компонент которого равен среднему арифметическому значению соответствующих компонентов заданных файлов (количество компонентов в исходных файлах одинаковое).
7. Даны два файла с числами. Получить новый файл, записав в него сначала все положительные числа из первого файла, потом все отрицательные числа из второго.
8. Даны два файла с числами. Получить новый файл, записав в него сначала все четные числа из первого файла, потом все нечетные числа из второго.
9. Даны два файла с числами. Получить новый файл, в котором на четных местах будут стоять компоненты, которые стоят на четных местах в первом файле, а на нечетных – компоненты, которые стоят на нечетных во втором (количество компонентов в исходных файлах одинаковое).
10. Дан файл f, компонентами которого являются символы. Переписать в файл g все знаки препинания файла f, а в файл h – все остальные символы файла f.
11. Дан файл f, элементами которого являются символы. Переписать в файл g все цифры файла f, а в файл h – все остальные символы файла f.
12. Дан два файла с одинаковым количеством компонент, компонентами которых являются символы. Выяснить, совпадают ли попарно их компоненты. Если нет, получить номер первого элемента, в котором эти файлы отличаются.
13. Дан файл, компонентами которого являются целые числа. Переписать в новый файл сначала все отрицательные компоненты из первого, потом все положительные.
14. Дан файл, компонентами которого являются символы. Создать новый файл таким образом, чтобы на четных местах у него стояли компоненты, стоящие на нечетных в первом файле, и наоборот.

15. Дан файл, компонентами которого являются числа. Число компонент файла делится на два. Создать новый файл, в который будет записываться наименьшее из каждой пары чисел первого файла.
16. Дан файл, компонентами которого являются числа. Число компонент файла делится на два. Создать новый файл, в который будет записываться среднее арифметическое из каждой пары чисел первого файла.
17. Дан файл, компонентами которого являются символы. Переписать все символы в новый файл в обратном порядке.
18. Даны два файла с одинаковым количеством компонент, компонентами которых являются натуральные числа. Создать новый файл, в который будут записываться числа по следующему правилу. Берется первое число из первого файла и первое из второго. Если одно из них делится нацело на другое, то их частное записывается в новый файл. Затем берется второе число из первого и второе число из второго и т.д.
19. Дан файл, компонентами которого являются символы. Переписать в новый файл все символы, которым в первом файле предшествует данная буква.
20. Дан файл, компонентами которого являются символы. Переписать в новый файл все символы, за которыми в первом файле следует данная буква.

III. Работа с двоичными файлами.

1. Создать файл и записать в него квадраты натуральных чисел от 1 до n. Вывести на экран все компоненты файла с нечетным порядковым номером.
2. Создать файл и записать в него степени числа 3. Вывести на экран все компоненты файла с четным порядковым номером.
3. Создать файл и записать в него обратные натуральные числа $1, \frac{1}{2}, \ldots, \frac{1}{n}$. Вывести на экран все компоненты файла с порядковым номером, кратным 3.
4. Создать файл и записать в него n первых членов последовательности Фибоначчи. Вывести на экран все компоненты файла с порядковым номером, не кратным 3.
5. Создать файл, состоящий из n целых чисел. Вывести на экран все четные числа данного файла.
6. Создать файл, состоящий из n целых чисел. Вывести на экран все отрицательные числа данного файла.
7. Создать файл, состоящий из n целых чисел. Вывести на экран все числа данного файла, попадающие в заданный интервал.
8. Создать файл, состоящий из n целых чисел. Вывести на экран все числа данного файла, не попадающие в заданный интервал.
9. Создать файл, состоящий из n целых чисел. Вывести на экран все числа данного файла, не кратные заданному числу.
10. Создать файл, состоящий из n вещественных чисел. Вывести на экран все числа данного файла, не попадающие в данный диапазон.
11. Создать файл, состоящий из n вещественных чисел. Вывести на экран все числа данного файла с нечетными порядковыми номерами, большие заданного числа.
12. Создать файл, состоящий из n вещественных чисел. Вывести на экран все числа данного файла с четными порядковыми номерами, меньшие заданного числа.

13. Создать файл, состоящий из n вещественных чисел. Найти сумму всех положительных чисел данного файла.
14. Создать файл, состоящий из n вещественных чисел. Подсчитать среднее арифметическое чисел файла, стоящих на четных позициях.
15. Создать файл, состоящий из n вещественных чисел. Найти максимальное значение среди чисел файла, стоящих на нечетных позициях.

4.4. Самостоятельная работа

Задача 1. N человек играли в карточную игру. После каждой раздачи был один выигравший и один проигравший. Они сыграли M раздач и каждый раз записывали, кто у кого сколько выиграл (в одной раздаче нельзя выиграть больше 100 очков). Теперь игроки хотят узнать, сколько каждый из них выиграл. Помогите им.

Во входном текстовом файле в первой строке находится натуральное число N ($1 \le N \le 50$). В следующих N строках записаны имена игроков. Длина имени не превосходит 10 символов и не содержит пробелов. В следующей строке записано целое число M ($0 \le M \le 100$). Далее записаны M строк с результатами раздач в формате:

<имя выигравшего игрока> <имя проигравшего игрока> <выигрыш>.

В выходной файл необходимо записать N строк в формате:

<имя игрока> <выигрыш>

Если игрок проиграл, то его выигрыш отрицательный. Все имена и числа в одной строке разделяются единственным пробелом. Строки располагаются в файле в произвольном порядке. Например:

Input.txt	*Output.txt*
3 Петя Вася Саша 3 Петя Саша 10 Петя Вася 20 Саша Вася 10	Петя 30 Саша 0 Вася -30

Задача 2. В сообщении могут встречаться номера телефонов, записанные в формате xx-xx-xx, xxx-xxx или xxx-xx-xx. Найти все номера телефонов, содержащиеся в сообщении.

Исходное сообщение содержится в текстовом файле input.txt. Номера телефонов следует вывести в файл output.txt, при этом каждый номер выводится с новой строки. Например:

input.txt	*output.txt*
У Васи телефон 12-34-56. А с Петей можно связаться по номеру 789-012.	12-34-56 786-012

Задача 3. В сообщении могут содержаться даты в формате дд.мм.гг. Найти все даты, содержащиеся в сообщении.

Исходное сообщение содержится в текстовом файле input.txt. Даты следует вывести в файл output.txt, при этом каждая дата выводится с новой строки и через пробел указывается дата предшествующего дня. Например:

input.txt	*output.txt*
У Васи день рождения 01.01.84. А с Петей я	01.01.84 31.12.83
познакомился 01.12.95. Встреча друзей	01.12.95 30.11.94
назначена на 25.07.09.	25.07.09 24.07.09

Задача 4. В сообщении может содержаться время в формате чч:мм:сс. Преобразовать каждое время к формату чч:мм, применив правило округления до целого числа минут и вывести на экран.

Исходное сообщение содержится в текстовом файле input.txt. Время следует вывести в файл output.txt, при этом каждое время выводится с новой строки. Например:

input.txt	*output.txt*
Вася потратил 01:24:34 на подготовку к	01:25
контрольной работе. А Петя 01:59:12 и	01:59
лег спать в 23:59:59.	00:00

Задача 5. В сообщении могут содержаться IP-адреса компьютеров в формате d.d.d.d, где d целое число из диапазона от 0 до 255. Найти все IP-адреса, содержащиеся в сообщении.

Исходное сообщение содержится в текстовом файле input.txt. Время следует вывести в файл output.txt, при этом каждый IP-адрес выводится с новой строки. Например:

input.txt	*output.txt*
У моего компьютера IP-адрес 127.23.3.78.	127.23.3.78
А у Пети то ли 127.23.3.258, то ли 127.23.32.58	127.23.32.58

5. СТРУКТУРЫ

5.1. Общие сведения

Структуры, как и массивы, относятся к составным типам данных. Однако в отличие от массива, элементы которого однотипны, структура может содержать элементы разных типов. Синтаксис описания структуры:

```
struct <имя структуры>
    {
      <тип 1> <идентификатор 1>;
      <тип 2> <идентификатор 2>;
       ...
      <тип n> <идентификатор n>;
    };
```

Ключевое слово struct используется для определения типа данных структура. Идентификаторы, объявленные внутри фигурных скобок, называются членами-данными структуры или полями. Например, рассмотрим следующее описание структуры:

```
struct sotrudnik
      {
        string familiya,imya,otchestvo, adres;
        int nomer;
      };
```

Здесь определяется структура sotrudnik, имеющая пять членов-данных, четыре из которых имеют тип string и один – тип int.

После определения структуры ее тип может использоваться так же, как и любой из стандартных типов данных int, float и т.д. Например:

```
sotrudnik sotr;      //описание переменной sort
//описанием одномерного массива, где каждый элемент имеет тип sotrudnik
sotrudnik mas[10];
sotrudnik *p;        //описание указателя
```

Можно пропустить шаг первый, и описать переменную, указатель или массив, где базовым элементом является тип sotrudnik, следующим образом:

```
struct sotrudnik
       {
        string familiya,imya,otchestvo, adres;
        int nomer;
       } sort, *p, mas[10];
```

В данном случае переменная, указатель и массив описываются в момент определения самой структуры.

При описании переменной типа структура можно проводить ее инициализацию, для этого значения ее элементов перечисляются в фигурных скобках в порядке их описания. Например, следующим образом:

```
sotrudnik sort={"Ivanov", "Ivan", "Ivanovich", "Saratov", "123456"};
```

Или

```
struct sotrudnik
       {
        string familiya,imya,otchestvo, adres;
        int nomer;
       } sort={"Ivanov", "Ivan", "Ivanovich", "Saratov", "123456"};
```

Доступ к полям структуры выполняется с помощью операций выбора: при обращении к полю через имя переменной используется операция . (точка), при обращении через указатель используется операция ->. Например:

```
sotr.familiya="Ivanov";
mas[0].imya="Ivan";
p->otchectvo="Ivanovich";
```

В программе переменные типа структура могут использоваться как переменные любых других типов. Например:

```
//ввод данных в переменную sort
cin>>sotr.familiya;
cin>> sotr.imya;
cin>> sotr.otchestvo;
```

```
cin>> sotr.adres;
cin>> sotr.nomer;

//вывод данных связанных с указателем p
cout <<p->familiya <<'\t' <<p->imya <<'\t' <<p->otchestvo <<'\t' <<p->sotr.adres <<endl;

//обработка массива
for (int i=0; i<n; i++)
   if (mas[i]. familiya==poisk) cout <<mas[i].adres;
```

Структуры языку C++ "достались в наследство" от языка C. В C++ определение структуры расширилось за счет включения в нее членов-функций, в том числе, конструкторов и деструкторов. Рассмотрим расширенное описание структуры:

```
struct <имя структуры>
    {
      // открытые члены – данные и функции
      ...
      private:
      // закрытые члены структуры – данные и функции
      ...
     };
```

Обратите внимание на введение нового ключевого слова – private, которое сообщает компилятору о том, что следующие за ним члены структуры являются закрытыми (доступ к ним возможен только из данной структуры).

Рассмотрим следующий пример:

```
#include "iostream"
#include "cmath"
using namespace std;
struct point //описание структуры
{
  int x, y;      // открытые члены-данных
  void show();       //открытый член-функция
  private:
  double dlina();    //закрытый член-функция
};
double point::dlina() //реализация члена функции dlina
{
 return sqrt((double)(x*x+y*y));
}
void point::show() //реализация члена функции show
{
  cout<<"Координаты точки:("<<x<<", "<<y<<")"<<endl;
  double d= dlina(); //допустимое обращение к закрытому члену структуры
  cout<<"Расстояние до начала координат: "<<d<<endl;
  cout<<endl;
}
```

```
int main()
{
 point a,b;  //описание переменных типа point
 a.x=1; a.y=1;      //обращение к отрытым членам-данных структуры point
 a.show();          //обращение к отрытому члену-функции структуры point
 // Ошибка! Недопустимо обращаться к закрытому члену структуры из
 //произвольного места программы.
 // double z=a.dlina();
 b.x=4; b.y=3;      //обращение к отрытым членам-данных структуры point
 b.show();          //обращение к отрытому члену-функции структуры point
 return 0;
}
```

Структура наряду с другими членами может содержать конструктор, который используется для инициализации членов-данных структуры. Добавим в предыдущий пример конструктор:

```
#include "iostream"
#include "cmath"
using namespace std;
struct point //описание структуры
{
 int x, y;     // открытые члены-данных
 point(int a, int b); //конструктор
 void show();       //открытый член-функция
 private:
 double dlina();    //закрытый член-функция
};
point::point(int a, int b) //реализация конструктора
{
   x=a;y=b;
}
double point::dlina() //реализация члена-функции dlina
{
 return sqrt((double)(x*x+y*y));
}
void point::show() //реализация члена функции show
{
   cout<<"Координаты точки:("<<x<<", "<<y<<")"<<endl;
   double d= dlina();
   cout<<"Расстояние до начала координат: "<<d<<endl;
   cout<<endl;
}
int main()
{
 point *a= new point(1,1) ;      //вызов конструктора
 a->show();         //обращение к отрытому члену-функции структуры point
 point *b; //описание указателя на тип point
 //ошибка: т.к. b это только указатель тип point и еще не производилось
```

```
//выделение памяти под данный объект, следовательно, не возможно
//обратиться к его членами
b->x=3; b->y=4;
//устанавливаем указатель на b на область памяти, связанную с указателем a
b=a;
b->show();
b->x=100;
a->show();
b->show();
return 0;
}
```

Имя конструктора совпадает с именем структуры. Для конструктора не указывается тип возвращаемого значения, т.к. назначение конструктора инициализировать члены-данные. В качестве параметров конструктору передаются аргументы, которые будет использоваться при инициализации.

Если структура содержит конструктор, то работа с ней возможна только через указатели. При описании указателя на тип структура можно сразу вызвать конструктор с помощью операции new. Например, в нашем примере имеется строка:

```
point *a= new point(1,1);
```

где *a – это указатель на тип структура, а после операции new мы указываем имя конструктора и в круглых скобках перечисляем значения параметров.

Так как a является указателем, то для обращения к полям структуры нужно использовать операцию выбора ->. Например, чтобы обратиться к полю x мы записываем a->x.

После того как была выполнена операция присваивания b=a, a и b стали указывать на одну и ту же область памяти. Теперь если изменить значение по указателю b, то обратившись к члену-данных show для a и b мы получим одинаковые результаты. Этот факт нужно учитывать при работе с указателями.

Замечание. В C++ тип данных структура получил свое дальнейшее развитие - появились классы. Синтаксис класса очень похож на синтаксис структуры, хотя вместо ключевого слова struct используется ключевое слово class, а также по умолчанию в структуре все члены открыты, а в классе закрыты. Появление классов дало развитие новой технологии программирования – объектно-ориентированного программировании, реализующих такие механизмы как наследование, полиморфизм, инкапсуляция. Таким образом, структуры – это переход от структурного программирования к ООП.

5.2. Примеры решения задач

1. Окружность в пространстве задана своим центром и радиусом. Определить, какое количество точек заданного в пространстве множества лежит на окружности.

```
#include "iostream"
#include "cmath"
using namespace std;
struct point //структура для хранения координат точки в пространстве
{
  int x, y, z;
};
```

```
//функция, вычисляющая расстояние между двумя точками в пространстве
double dlina(point a, point b)
{
  return sqrt(pow(a.x-b.x,2)+pow(a.y-b.y,2)+pow(a.z-b.z,2));
}

int main()
{
 point circle, a[10];
 double r;
 int n;
 cout<<"Введите центр окружности: ";
 cin >>circle.x>>circle.y>>circle.z;
 cout<<"Введите радиус окружности: ";
 cin >>r;
 cout<<"Введите количество точек в множестве: ";
 cin >>n;
 for(int i=0; i<n; i++) //ввод координат точек заданного множества
 {
        cout<<"Введите координаты " << i<<"-той точки:";
        cin>>a[i].x>>a[i].y>>a[i].z;
 }
 int k=0;
 for (int i=0; i<n; i++)
  if (dlina(circle,a[i])==r) //проверяем, лежит ли точка на заданной окружности
    k++;
 cout<<"Количество точек множества, лежащих на заданной окружности,  = "<< k;
 return 0;
}
```

2. Дан файл input.txt, в котором содержится библиотечная ведомость. Каждая строка ведомости содержит следующую информацию: библиотечный номер, фамилия автора, название книги, год издания. В файл output.txt вывести данную библиотечную ведомость, удалив из нее книгу с заданным номером.

```
#include <fstream>
#include <iostream>
#include <iomanip>
using namespace std;

//открываем глобальные файловые  потоки
ifstream  in("input.txt");
ofstream  out("output.txt");

//описание структуры
struct bibl
   {
     int number,year;
     char family[20],nazv[50];
     void show(); //член-функция для вывода информации на экран
     void print();  //член-функция для вывода информации в файл
   };
```

```
void bibl::show()
{
  cout <<setw(8)<<number <<setw(10)<< family <<setw(8)<<nazv <<setw(8)<< year
       << endl;
}
void bibl::print()
{
  out <<setw(8)<<number <<setw(10)<< family <<setw(8)<<nazv <<setw(8)<<year
      <<endl;
}
int main()
{
   bibl book[10]; //описываем массив структур
   int num;
   int i,j,n=0;
    if(!in) cout<<"Ошибка при открытии файла input.txt\n";
     else
     {
       while(in.peek()!=EOF) //чтение данных из файла
         {
           in >>book[n].number;
           in >>book[n].family;
           in >>book[n].nazv;
           in >>book[n].year;
           book[n].show(); //вывод прочтенных данных на экран через член-функцию
           n++;
         };
     cout<<"Введите номер книги, которую\n";
     cout<<"нужно убрать из списка:";
     cin>>num;
     if(!out) cout<<"Ошибка при открытии файла output.txt\n";
      else
      {
        j=0;
       for(i=0;i<n && !j;i++)
       //поиск структуры по совпадению поля number с num для удаления
       if(book[i].number==num)
         {
          //выполняем сдвиг в массиве на одну позицию вправо
           for(j=i;j<n-1;j++) book[j]=book[j+1];
           n--; //уменьшаем количество книг на 1 после удаления
         }
      //записываем результата в новый файл
        for(i=0;i<n;i++)
          book[i].print();//вывод данных на файл через член-функцию структуры
      }
 }
in.close();out.close(); //закрываем потоки
return 0;
```

}

input.txt	*номер книги*	*output.txt*
1 Ivanov C++ 2008	2	1 Ivanov C++ 2008
2 Petrov Java 2007		3 Petrov C/C++ 2008
3 Petrov C/C++ 2008		4 Ivanov C# 2009
4 Ivanov C# 2009		

5.3. Упражнения

I. Решить задачу, используя структуру point для хранения координат точки:

Замечание. В задачах с четными номерами множество точек задано на плоскости, в задачах с нечетными номерами множество точек задано в пространстве.

1-2. Найти точку, которая наиболее удалена от начала координат.
3-4. Найти точку, которая наименее удалена от начала координат.
5-6. Найти две наиболее удаленных друг от друга точки.
7-8. Найти две наиболее близко расположенных друг к другу точки.
9-10. Найти такую точку, что шар радиуса R с центром в этой точке содержит максимальное число точек заданного множества.
11-12. Найти такую точку, что шар радиуса R с центром в этой точке содержит минимальное число точек заданного множества.
13-14. Найти такую точку, сумма расстояний от которой до остальных точек множества максимальна.
15-16. Найти такую точку, сумма расстояний от которой до остальных точек множества минимальна.
17-18. Найти три различные точки из заданного множества точек, образующих треугольник наибольшего периметра.
19-20. Найти три различные точки из заданного множества точек, образующих треугольник наименьшего периметра.

II. Решить задачу, используя структуру содержащую члены-данные и члены-функции.

Замечание. Во всех задачах данного раздела подразумевается, что исходная информация хранится в текстовом файле input.txt, каждая строка которого содержит полную информацию о некотором объекте, результирующая информация должна быть записана в файл output.txt.

1. На основе данных входного файла составить список студентов группы, включив следующие данные: ФИО, год рождения, домашний адрес, какую школу окончил. Вывести в новый файл информацию о студентах, окончивших заданную школу.
2. На основе данных входного файла составить список студентов группы, включив следующие данные: ФИО, год рождения, домашний адрес, какую школу окончил. Вывести в новый файл список студентов, удалив из него студентов окончивших школу в текущем году.
3. На основе данных входного файла составить список студентов группы, включив следующие данные: ФИО, номер группы, результаты сдачи трех экзаменов. Вывести в новый файл информацию о студентах, успешно сдавших сессию.
4. На основе данных входного файла составить список студентов группы, включив следующие данные: ФИО, номер группы, результаты сдачи трех экзаменов.

Вывести в новый файл список студентов, удалив из него информацию о студентах «проваливших» сессию.

5. На основе данных входного файла составить багажную ведомость камеры хранения, включив следующие данные: ФИО пассажира, количество вещей, общий вес вещей. Вывести в новый файл информацию о тех пассажирах, средний вес багажа которых превышает заданный.
6. На основе данных входного файла составить багажную ведомость камеры хранения, включив следующие данные: ФИО пассажира, количество вещей, общий вес вещей. Вывести в новый файл багажную ведомость, удалив из нее информацию о тех пассажирах, средний вес багажа которых меньше заданного.
7. На основе данных входного файла составить автомобильную ведомость, включив следующие данные: марка автомобиля, номер автомобиля, фамилия его владельца, год приобретения, пробег. Вывести в новый файл информацию об автомобилях, выпущенных ранее определенного года.
8. На основе данных входного файла составить автомобильную ведомость, включив следующие данные: марка автомобиля, номер автомобиля, фамилия его владельца, год приобретения, пробег. Вывести в новый файл автомобильную ведомость, удалив из нее информацию об автомобилях, пробег которых менее заданного значения.
9. На основе данных входного файла составить список сотрудников учреждения, включив следующие данные: ФИО, год принятия на работу, должность, зарплата, рабочий стаж. Вывести в новый файл информацию о сотрудниках, имеющих зарплату ниже определенного уровня.
10. На основе данных входного файла составить список сотрудников учреждения, включив следующие данные: ФИО, год принятия на работу, должность, зарплата, рабочий стаж. Вывести в новый файл список сотрудников учреждения, удалив из него информацию о сотрудниках, принятых на работу в текущем году.
11. На основе данных входного файла составить инвентарную ведомость склада, включив следующие данные: вид продукции, стоимость, сорт, количество. Вывести в новый файл информацию о той продукции, количество которой менее заданной величины.
12. На основе данных входного файла составить инвентарную ведомость склада, включив следующие данные: вид продукции, стоимость, сорт, количество. Вывести в новый файл инвентарную ведомость склада, увеличив стоимость каждого вида продукции на x%.
13. На основе данных входного файла составить инвентарную ведомость игрушек, включив следующие данные: название игрушки, ее стоимость (в руб.), возрастные границы детей, для которых предназначена игрушка. Вывести в новый файл информацию о тех игрушках, которые предназначены для детей от N до M лет.
14. На основе данных входного файла составить инвентарную ведомость игрушек, включив следующие данные: название игрушки, ее стоимость (в руб.), возрастные границы детей, для которых предназначена игрушка. Вывести в новый файл инвентарную ведомость игрушек, уменьшив стоимость каждого вида игрушек на x%.

15. На основе данных входного файла составить список вкладчиков банка, включив следующие данные: ФИО, № счета, сумма, год открытия счета. Вывести в новый файл информацию о тех вкладчиках, которые открыли вклад в текущем году.
16. На основе данных входного файла составить список вкладчиков банка, включив следующие данные: ФИО, № счета, сумма, год открытия счета. Вывести в новый файл информацию о тех вкладчиках, сумма вклада которых превышает заданное значение.
17. На основе данных входного файла составить список студентов, включающий фамилию, факультет, курс, группу, 5 оценок. Вывести в новый файл информацию о тех студентах, которые сдали сессию на 4 и 5.
18. На основе данных входного файла составить список студентов, включающий фамилию, факультет, курс, группу, 5 оценок. Вывести в новый файл информацию о тех студентах, которые имеют хотя бы одну двойку.
19. На основе данных входного файла составить список студентов, включающий ФИО, курс, группу, результат забега. Вывести в новый файл информацию о студентах заданной группы.
20. На основе данных входного файла составить список студентов, включающий ФИО, курс, группу, результат забега. Вывести в новый файл список студентов, удалив из него информацию о тех студентах, которые не выполнили норматив по бегу.

5.4. Самостоятельная работа

Задача 1. Дано множество точек на плоскости. Выбрать из них такие четыре точки, которые составляют квадрат наибольшего периметра.

Исходные данные хранятся в файле input.txt, при этом координаты каждой точки записаны на отдельной строке через пробел. В файл output.txt вначале выводятся координаты найденных точек. Например:

input.txt	*output.txt*
-1 1	-1 1
1 1	1 3
3 1	3 1
1 3	1 -1
1 4	
1 -1	
1 -2	

Задача 2. Определить радиус и центр окружности, проходящей через три различные точки заданного множества точек на плоскости и содержащей внутри себя наибольшее количество точек этого множества.

Исходные данные хранятся в файле input.txt, при этом координаты каждой точки записаны на отдельной строке через пробел. В файл output.txt вначале выводятся координаты центра, а затем с новой строки радиус найденной окружности. Например:

input.txt	output.txt
-3 0 -2 0 2 0 3 0 -1 1 1 1 0 3 3 3 0 -1 -1 -2	0 0 3

6. СОРТИРОВКИ

Сортировкой или упорядочиванием списка объектов называется расположение этих объектов по возрастанию или убыванию согласно определенному линейному отношению порядка, такому, например, как отношение «≤» для целых чисел.

Будем далее предполагать, что сортируемые объекты являются записями, содержащими одно или несколько полей. Одно из полей, называемое ключом, имеет такой тип данных, что на нем определено отношение линейного порядка «≤». Например, это может быть целое или действительное число, строка. Задача сортировки состоит в упорядочивании записей таким образом, чтобы значения ключевого поля составляли неубывающую последовательность. Другими словами, записи $r_1, r_2, \ldots, r_n$ со значениями ключей $k_1, k_2, \ldots, k_n$ (не обязательно различными) надо расположить в порядке $r_{i_1}, r_{i_2}, \ldots, r_{i_n}$ таком, что $k_{i_1} \le k_{i_2} \le \ldots \le k_{i_n}$.

Существуют различные критерии оценки времени выполнения алгоритмов сортировки. Первой и наиболее общей мерой времени выполнения является количество шагов алгоритма, необходимых для упорядочивания n записей. Если размер записей большой, то перестановка записей занимает больше времени, чем все другие операции. Поэтому другой общей мерой служит количество перестановок записей, выполненных в ходе алгоритма.

В приведенных далее листингах программ будем использовать следующие обозначения: a – массив из n записей; key – одно из полей записей, которое является ключом; temp – переменная того же типа, что и элементы a. Каждый вид сортировки будет оформлен в виде отдельной функции.

6.1. Метод «пузырька»

Представим, что записи, подлежащие сортировке, хранятся в массиве, расположенном вертикально. Записи с малыми значениями ключевого поля более «легкие» и «всплывают» вверх наподобие пузырька. При первом проходе вдоль массива, начиная проход снизу, берется первая запись массива, и ее ключ поочередно сравнивается с ключами последующих записей. Если встречается запись с более «тяжелым» ключом, то эти записи меняются местами. При встрече с записью с более «легким» ключом эта запись становится эталоном для сравнения, и все последующие записи сравниваются с этим новым, более «легким» ключом. В

результате запись с наименьшим значением ключа оказывается в самом верху массива. Во время второго прохода вдоль массива находится запись со вторым по величине ключом, которая помещается под записью, найденной при первом проходе массива, т.е. на вторую сверху позицию, и т.д. Отметим, что во время второго и последующих проходов вдоль массива нет необходимости просматривать записи, найденные за предыдущие проходы, так как они имеют ключи, меньшие, чем у оставшихся записей. Другими словами, во время i-ого прохода не проверяются записи, стоящие на позициях выше i.

```
void sort(mas *a, int n)
{
 mas temp;
 int i, j;
 for(i=0; i<n-1; i++)
  for (j=n-1; j>i; j--)
   if (a[j].key<a[j-1].key)
   {
      temp=a[j];
      a[j]=a[j-1];
      a[j-1]=temp; }
   }
```

Замечание. Напомним, что в C++ нумерация элементов массива начинается с 0, поэтому в массиве из n элементов последний элемент будет с номером n-1.

Пример. Рассмотрим список студентов:

ФИО	*Год рождения*
Петров	1985
Эдуардов	1983
Кузнецова	1984
Антонов	1982
Семенова	1981
Власов	1983

Для этого примера объявление типов данных будет выглядеть следующим образом:

```
struct mas
       {
         int key;          // ключевое поле
         char fio[30];     // информационное поле
        };
mas *a;     // указатель на нулевой элемент массив
mas temp;  // рабочая переменная, которая будет использоваться для
           // перестановки местами двух элементов массива
```

Применим алгоритм «пузырька» для упорядочивания списка студентов по возрастанию года рождения. В таблице показаны 5 проходов алгоритма (n=6). Линии указывают позицию, выше которой записи уже упорядочены. После пятого прохода все записи, кроме последней, стоят на нужных местах. Но последняя запись не случайно оказалась последней: она также уже стоит на нужном месте. Поэтому сортировка закончена.

Начальное положение	1-й проход	2-й проход	3-й проход	4-й проход	5-й проход
1985	1981	1981	1981	1981	1981
1983	1985	1982	1982	1982	1982
1984	1983	1985	1983	1983	1983
1982	1985	1983	1985	1983	1983
1981	1982	1984	1983	1985	1984
1983	1983	1983	1984	1984	1985

6.2. Сортировка вставками

Идея этого метода заключается в том, что на i-ом этапе мы «вставляем» элемент a[i] в нужную позицию среди элементов a[1], a[2],..., a[i-1], которые уже упорядочены. Чтобы сделать процесс перемещения элемента a[i] более простым, полезно ввести элемент a[0], чье значение ключа будет меньше значения ключа любого элемента a[1], a[2],..., a[n]. Обозначим ключевое значение элемента a[0] символом -∞. Если такое значение нельзя применить, то при вставке a[i] в позицию j-1 нужно проверить, не будет ли j=1. Если нет, тогда сравнивать элемент a[i], который находится в позиции j с элементом a[j-1].

```
void sort(mas *a, int n)
{
 mas temp;
 int i,j;
 for( i=2; i<=n; i++)
  {
   j=i;
   while (a[j].key<a[j-1].key)
    {
     temp=a[j];
     a[j]=a[j-1];
     a[j-1]=temp;
     j--; }
  }
}
```

Замечание. При заполнении массива в поле key нулевого элемента было записано значение –MAXINT (<values.h>), как того требовал алгоритм. А n записей были помещены в массив начиная с первого элемента. Поэтому для выполнения алгоритма сортировки вставками потребуется одномерный массив размерностью n+1.

Пример. В таблице показаны этапы алгоритма сортировки вставками, который используется для упорядочивания списка из предыдущего примера. После каждого этапа алгоритма элементы, расположенные выше линии, уже упорядочены, хотя между ними на последующих этапах могут быть вставлены элементы, которые на данном этапе расположены ниже линии.

Начальное положение	1-й проход	2-й проход	3-й проход	4-й проход	5-й проход
-∞	-∞	-∞	-∞	-∞	-∞
1985	1983	1983	1982	1981	1981
1983	1985	1984	1983	1982	1982
1984	1984	1985	1984	1983	1983
1982	1982	1982	1985	1984	1983
1981	1981	1981	1981	1985	1984
1983	1983	1983	1983	1983	1985

6.3. Сортировка посредством выбора

На i-ом этапе сортировки выбирается запись с наименьшим ключом среди записей a[i],..., a[n] и меняется местами с записью a[i]. В результате после i-го этапа все записи a[1],..., a[i] будут упорядочены.

```
void sort(mas *a, int n)
{
 mas temp;
 int lowkey;        // текущий наименьший ключ, найденный при проходе по
                    //элементам a[i],...,a[n]
 int lowindex;            // позиция элемента с ключом lowkey
 int i, j;
 for( i=0; i<n-1; i++)
 {
   lowindex=i;  lowkey=a[i].key;
   for (j=i+1; j<n;j++)
      if (a[j].key<lowkey) {lowkey=a[j].key; lowindex=j;}
    temp=a[i];  a[i]=a[lowindex];   a[lowindex]=temp;
 }
}
```

Пример. В таблице показаны этапы сортировки посредством выбора для списка из примера пункта 6.1. Линия в таблице показывает, что элементы, расположенные выше нее, имеют наименьшие значения ключей и уже упорядочены. После (n-1)-го этапа элемент a[n] также стоит на «правильном» месте, т.к. выше него все записи имеют меньшие значения ключей.

Начальное положение	1-й проход	2-й проход	3-й проход	4-й проход	5-й проход
1985	1981	1981	1981	1981	1981
1983	1983	1982	1982	1982	1982
1984	1984	1984	1983	1983	1983
1982	1982	1983	1984	1983	1983
1981	1985	1985	1985	1985	1984
1983	1983	1983	1983	1984	1985

Замечание

Рассмотренные нами алгоритмы являются простыми схемами сортировки. Время работы этих алгоритмов пропорционально n^2 как в среднем, так и в худшем случае. Если же сравнивать эти алгоритмы с точки зрения количества перестановок, то более предпочтительным оказывается алгоритм выбора. Количество перестановок в этом алгоритме пропорционально n, в то время как в первых двух - n^2.

Для больших n простые алгоритмы сортировки заведомо проигрывают алгоритмам со временем выполнения пропорциональным $n \log n$. Значение n, начиная с которого быстрые

алгоритмы становятся предпочтительней, зависит от многих факторов. Быстрые алгоритмы более сложны в реализации и в данном пособии рассмотрены не будут.

Для небольших значений n рекомендуется применять простой в реализации алгоритм сортировки Шелла, который имеет временную сложность $O(n^{1,5})$. Этот алгоритм является обобщением алгоритма «пузырька».

6.4. Алгоритм сортировки Шелла

Массив a из n элементов упорядочивается следующим образом. На первом шаге упорядочиваются элементы n/2 пар (a[i], a[n/2+i]) для 1≤i≤n/2; на втором шаге упорядочиваются элементы в n/4 группах из четырех элементов (a[i],a[n/4+i],a[n/2+i],a[3n/4+i]) для 1≤i≤n/4; на третьем шаге упорядочиваются элементы в n/8 группах из восьми элементов и т.д. На последнем шаге упорядочиваются элементы сразу во всем массиве a. На каждом шаге для упорядочивания элементов используется метод сортировки вставками.

```
void sort(mas *a, int n)
{
 mas temp;
 int i, j, incr=n/2;
 while (incr>0)
  {
   for( i=incr; i<n; i++)
   {
     j=i-incr;
     while (j>=0)
      if (a[j].key>a[j+incr].key)
          { temp=a[j]; a[j]=a[j+incr]; a[j+incr]=temp; j=j-incr;}
       else j= -1;
   }
   incr= incr/2;
  }
}
```

Пример. Рассмотрим этапы работы алгоритма сортировки Шелла для следующего примера: упорядочить по неубыванию следующие числа: 1, 7, 3, 2, 0, 5, 0, 8.

Начальное положение	1 шаг	2 шаг	3 шаг
1	0	0	0
7	5	2	0
3	0	0	1
2	2	5	2
0	1	1	3
5	7	7	5
0	3	3	7
8	8	8	8

На первом этапе рассматриваются следующие пары: 0-й и 4-й элементы со значением 1 и 0 соответственно, 1-й и 5-й элементы со значением 7 и 5, 2-й и 6-й элементы со значением 3 и 0, 3-й и 7-й элементы со значением 2 и 8. Всего 4 пары по 2 элемента (n=8). В каждой паре упорядочиваем элементы в порядке возрастания

методом вставки. В данном случае, в первых трех парах элементы поменяются местами, в четвертой все останется по-прежнему. В результате получим пары (0,1), (5,7), (0,3), (2,8) (см. столбец 2 таблицы). На втором этапе рассматриваются четверки: 0-й, 2-й, 4-й и 6-й элементы со значениями 0, 0, 1, 3 соответственно, 1-й, 3-й, 5-й и 7-й элементы со значениями 5, 2, 7, 8. После упорядочивания получим четверки: (0,0,1,3), (2,5,7,8). На последнем этапе рассматривается весь массив: (0,2,0,5,1,7,3,8). После упорядочивания получаем (0,0,1,2,3,5,7,8) – окончательный результат.

Замечание. В листинге алгоритма Шелла использовалась убывающая последовательность шагов n/2, n/4, n/8, …, 2, 1. В общем случае алгоритм работает с любой убывающей последовательностью шагов, у которой последний шаг равен 1. Например, при n, кратном 3, можно использовать последовательность n/3, n/9, n/27, …, 3, 1.

6.5. Решение практических задач с использованием различных алгоритмов сортировок

1. В файле input.txt содержатся сведения о группе студентов в формате:

номер группы;
количество студентов в группе;
запись о каждом студенте группы содержит следующие сведения: фамилия, имя, отчество, оценки по пяти предметам.

Переписать данные файла input.txt в файл output.txt, отсортировав их по убыванию средней оценки. Для каждого студента вычисленную среднюю оценку вывести в файл output.txt.

```
#include "fstream"
#include "string"
#include "iostream"
#include "iomanip"
using namespace std;
//открываем глобальные файловые  потоки
ifstream  in("input.txt");
ofstream  out("output.txt");
struct mas
    {
     string fam, name, secondname;      // фамилия, имя, отчество
     int ses[5];                        // оценки по пяти предметам
     double key;                        // средняя оценка
     void print();
    };
void mas::print() //вывод данных в выходной поток
{
  out <<setw(12)<<left<<fam <<setw(10)<< name <<setw(15)<<secondname ;
  for (int i=0; i<5; i++)
    out<<setw(3)<<ses[i];
  out<<setw(5)<<key<<endl;
}
// сортировка массива записей из n элементов методом «пузырька»
```

```
void sort(mas *a, int n)
{mas temp;
 int i,j;
 for(i=0; i<n-1; i++)
 for (j=n-1; j>i; j--)
   if (a[j].key>a[j-1].key) {temp=a[j]; a[j]=a[j-1]; a[j-1]=temp;}
}
int main()
{
 int n=0,m, i;
 mas stud [20];
 if(!in) cout<<"Ошибка при открытии файла input.txt\n";
  else
   {
    in>>m;  // считываем номер группы
    while(in.peek()!=EOF) // считываем данные про всех студентов
     {
        in >>stud[n].fam;
        in >>stud[n].name;
        in >>stud[n].secondname;
        //считываем оценки и высчитываем средний балл
        stud[n].key=0;
        for (i=0; i<5;i++)
         {
           in >>stud[n].ses[i];
          stud[n].key+=stud[n].ses[i];
         }
        stud[n].key/=5;
        n++;
     }
   sort(stud,n);    // сортируем массива записей
   // выводим отсортированные данные в файл output.txt
   out<<m<<endl;
   for (i=0;i<n; i++)
     stud[i].print();
 }
 in.close();out.close();   // закрываем файлы
 return 0;
}
```

input.txt

```
111
Иванов Иван Иванович 5 4 5 4 5
Иванцов Сергей Петрович 5 4 4 5 4
Иванова Нина Юрьевна 5 5 5 5 5
Смирнова Анна Дмитриевна 3 4 3 3 3
Сидоров Андрей Григорьевич 4 3 4 3 4
```

output.txt

```
111
Иванова   Нина     Юрьевна      5 5 5 5 5 5
Иванов    Иван     Иванович     5 4 5 4 5 4.6
Иванцов   Сергей   Петрович     5 4 4 5 4 4.4
Сидоров   Андрей   Григорьевич  4 3 4 3 4 3.6
Смирнова  Анна     Дмитриевна   3 4 3 3 3 3.2
```

2. Дана матрица размерностью n×n, содержащая целые числа. Отсортировать каждую строчку матрицы по возрастанию элементов, используя алгоритм выбора.

```
#include "fstream"
#include "iostream"
#include "iomanip"
using namespace std;
ifstream  in("input.txt");
ofstream  out("output.txt");
// сортировка одномерного массива методом выбора
void sort(int *a, int n)
{int temp;
 int lowindex, lowkey, i, j;
 for( i=0; i<n-1; i++)
  { lowindex=i; lowkey=a[i];
    for (j=i+1; j<n;j++)  if (a[j]<lowkey)  {lowkey=a[j]; lowindex=j;}
    temp=a[i];
    a[i]=a[lowindex];
    a[lowindex]=temp;
    }
  }
int main()
{
  int n,m,i,j;
  int a[10][10];
  //ввод данных из файла input.txt
  in>>n >>m;
  for (i=0; i<n; i++)
    for (j=0; j<m; j++)
                in>>a[i][j];
  //сортируем каждую строку двумерного массива с помощью сортировки
  // методом выбора
  for (i=0; i<n; i++)  sort(a[i],m);
 // выводим обработанные данные в файл output.txt
 out<< n<<'\t'<< m<<'\n';
 for (i=0; i<n; i++)
  {
    for (j=0; j<m; j++)
      out<<setw(5)<<a[i][j];
    out<<'\n';
  }
  in.close();out.close();    // закрываем файлы
  return 0;
}
```

input.txt

```
4 5
23    54    65    -9    0
8     96    -4    -7    6
100   12    90    1     2
2     -1    4     -5    -12
```

output.txt

```
4 5
-9    0     23    54    65
-7    -4    6     8     96
1     2     12    90    100
-12   -5    -1    2     4
```

3. Дана матрица размерностью n×n, содержащая целые числа. Отсортировать каждый столбец матрицы по убыванию элементов методом вставки.

```
#include "fstream"
#include "iostream"
#include "iomanip"
using namespace std;
ifstream  in("input.txt");
ofstream  out("output.txt");
// сортировка одномерного массива методом выбора (по убыванию)
void sort(int *a, int n)
{int temp;
 int lowindex, lowkey, i, j;
 for( i=0; i<n-1; i++)
  { lowindex=i; lowkey=a[i];
    for (j=i+1; j<n;j++)  if (a[j]>lowkey)  {lowkey=a[j]; lowindex=j;}
    temp=a[i];
    a[i]=a[lowindex];
    a[lowindex]=temp;
   }
  }
int main()
{
 int n,m,i,j;
 int a[10][10];
 //ввод данных из файла
 in>>n >>m;
 for (i=0; i<n; i++)
   for (j=0; j<m; j++)
              in>>a[i][j];
 int b[10];
 //каждый столбец массива a копируем в массив b, сортируем массив b по
 // убыванию элементов методом выбора, затем копируем элементы массива b
 //в обрабатываемый столбец массива a
 for (j=0; j<m; j++)
 {
  for (i=0; i<n; i++) b[i]=a[i][j];
  sort(b,n);
  for (i=0; i<n; i++) a[i][j]=b[i];
 }
 // выводим обработанные данные в файл output.txt
 out<< n<<'\t'<< m<<'\n';
 for (i=0; i<n; i++)
  {
    for (j=0; j<m; j++)
      out<<setw(5)<<a[i][j];
    out<<'\n';
  }
```

```
 in.close();out.close();
 return 0;
}
```

input.txt

```
4 5
23   54   65   -9   0
8    96   -4   -7   6
100  12   90   1    2
2    -1   4    -5   -12
```

output.txt

```
4    5
100  96  90  1   6
23   54  65  -5  2
8    12  4   -7  0
2    -1  -4  -9  -12
```

6.6. Упражнения

I. В файле input.txt содержатся сведения о группе студентов в формате:

номер группы;
запись о каждом студенте группы содержит следующие сведения: фамилия, имя, отчество, год рождения, оценки по пяти предметам.

Переписать данные файла input.txt в файл output.txt, отсортировав их:

1. по убыванию средней оценки методом вставки (среднюю оценку вывести в файл output.txt для каждого студента);
2. по возрастанию средней оценки методом выбора (среднюю оценку вывести в файл output.txt для каждого студента);
3. по убыванию средней оценки алгоритмом Шелла (среднюю оценку вывести в файл output.txt для каждого студента);
4. по убыванию суммы оценок методом «пузырька» (сумму оценок вывести в файл output.txt для каждого студента);
5. по убыванию суммы оценок методом вставки (сумму оценок вывести в файл output.txt для каждого студента);
6. по возрастанию суммы оценок методом выбора (сумму оценок вывести в файл output.txt для каждого студента);
7. по убыванию суммы оценок алгоритмом Шелла (сумму оценок вывести в файл output.txt для каждого студента);
8. в алфавитном порядке по фамилии методом «пузырька»;
9. в алфавитном порядке по фамилии методом вставки;
10. в алфавитном порядке по фамилии методом выбора;
11. в алфавитном порядке по фамилии алгоритмом Шелла;
12. в алфавитном порядке по фамилии, а затем по возрастанию года рождения методом «пузырька»;
13. в алфавитном порядке по фамилии, а затем по убыванию года рождения методом вставки;
14. в алфавитном порядке по фамилии, а затем по возрастанию года рождения методом выбора;
15. в алфавитном порядке по фамилии, а затем по убыванию года рождения алгоритмом Шелла;
16. в алфавитном порядке по фамилии, имени, отчеству методом «пузырька»;
17. в алфавитном порядке по фамилии, имени, отчеству методом вставки;
18. в алфавитном порядке по фамилии, имени, отчеству методом выбора;

19. в алфавитном порядке по фамилии, имени, отчеству алгоритмом Шелла;
20. в алфавитном порядке по фамилии, имени, отчеству, а затем по убыванию года рождения алгоритмом Шелла;

II. Дана матрица размерностью n×n, содержащая целые числа. Отсортировать:

1. каждую строчку матрицы по убыванию элементов методом «пузырька»;
2. каждую строчку матрицы по убыванию элементов алгоритмом Шелла;
3. каждую строчку матрицы по убыванию элементов методом вставки;
4. каждый столбец матрицы по возрастанию элементов методом выбора;
5. каждый столбец матрицы по возрастанию элементов алгоритмом Шелла;
6. каждый столбец матрицы по возрастанию элементов методом «пузырька»;
7. диагонали матрицы, параллельные главной, по убыванию элементов методом вставки;
8. диагонали матрицы параллельные главной по убыванию элементов методом выбора;
9. диагонали матрицы параллельные главной по убыванию элементов алгоритмом Шелла;
10. диагонали матрицы параллельные главной по убыванию элементов методом «пузырька»;
11. диагонали матрицы параллельные побочной по возрастанию элементов методом выбора;
12. диагонали матрицы параллельные побочной по возрастанию элементов алгоритмом Шелла;
13. диагонали матрицы параллельные побочной по возрастанию элементов методом «пузырька»;
14. диагонали матрицы параллельные побочной по возрастанию элементов методом вставки;
15. каждый столбец матрицы с номером 2i по убыванию элементов, а с номером 2i+1 по возрастанию элементов методом «пузырька»;
16. каждый столбец матрицы с номером 2i по возрастанию элементов, а с номером 2i+1 по убыванию элементов методом вставки;
17. диагонали матрицы, расположенные выше главной, по убыванию элементов, а диагонали матрицы, расположенные ниже главной, по возрастанию элементов методом выбора;
18. диагонали матрицы, расположенные выше главной, по возрастанию элементов, а диагонали матрицы, расположенные ниже главной, по убыванию элементов алгоритмом Шелла;
19. диагонали матрицы, расположенные выше побочной, по убыванию элементов, а диагонали матрицы, расположенные ниже побочной, по возрастанию элементов методом выбора;
20. диагонали матрицы, расположенные выше побочной, по возрастанию элементов, а диагонали матрицы, расположенные ниже побочной, по убыванию элементов методом вставки.

6.7. Самостоятельная работа

I. Дана последовательность, состоящая из N целых чисел. Отсортировать ее, используя алгоритм:

1. быстрой сортировки;
2. пирамидальной сортировки;
3. «карманной сортировки»;
4. поразрядной сортировки;
5. сортировки подсчетом;
6. сортировки вычёрпыванием;
7. сортировки слиянием.

II. Предположим, что необходимо отсортировать список элементов, состоящий из уже упорядоченной последовательности элементов, которые следует за несколькими «случайными» элементами. Какой из рассмотренных в этой главе методов сортировки или изученных вами самостоятельно методов наиболее подходит для этого решения?

III. Алгоритм называется устойчивым, если он сохраняет исходный порядок следования элементов с одинаковыми элементами ключей. Какие из рассмотренных в этой главе методов сортировки или изученных вами самостоятельно методов являются устойчивыми?

7. КЛАСС-КОНТЕЙНЕР ВЕКТОР

В стандарт C++ входит библиотека стандартных шаблонов (STL – Standard Template Library), которая является достаточно мощным и удобным инструментом для хранения и обработки данных. Основными элементами библиотеки STL являются *контейнеры*, *итераторы* и *алгоритмы*.

Контейнер – это объект, предназначенный для хранения других объектов. Все объекты в контейнере имеют один тип. Контейнер является шаблоном класса, а шаблоны позволяют создать одно определение класса или функции, которое впоследствии можно применить для ряда типов. Таким образом, можно создать контейнер, содержащий строки, целые числа или объекты какого-то другого типа. Непосредственно для контейнеров определено небольшое количество операций, однако для работы с ними можно использовать множество дополнительных операций, реализованных в библиотеке алгоритмов STL.

В библиотеке стандартных шаблонов определены два вида контейнеров – последовательные и ассоциативные. В последовательных контейнерах элементы хранятся в порядке их поступления, и доступ к ним осуществляется по номеру позиции. В данной части пособия будет рассмотрен класс последовательного контейнера vector. В ассоциативных контейнерах доступ к элементам осуществляется с помощью ключей.

Итератор – это тип, позволяющий обращаться к хранимым в контейнере элементам, перемещаясь от одного к другому. Можно сказать, что итераторы по отношению к контейнерам играют роль индексов или указателей. В библиотеке тип итератора определен для каждого стандартного контейнера, а вот индексирование поддерживают не все контейнеры.

Алгоритмы, как уже говорилось выше, выполняют операции над элементами контейнеров.

7.1. Работа с векторами

Вектор – это тип данных, похожий на массив, служащий тем же целям, что и массив, но имеющий существенные отличия. Подобно массиву, вектор имеет базовый тип и содержит набор значений этого типа. Однако массив имеет фиксированную размерность (длину), которая не может ни увеличиваться, ни уменьшаться, а длина вектора может динамически изменяться. Кроме того, описание вектора и работа с ним отличаются от описания и работы с массивом.

Замечание. Для того чтобы использовать в программе вектора, к ней необходимо подключить заголовочный файл "vector".

Чтобы объявить объект типа vector необходимо указать его имя и базовый тип элементов:

```
vector <базовый_тип_элементов> имя_объекта;
```

Замечание. В данном описании объекта типа vector скобки <> являются обязательными элементами.

Например, объявить вектор, состоящий из целых чисел, можно следующим образом:

```
vector <int> iVec;
```

Замечание. Ключевое слово vector определяет не имя типа, а имя шаблона, который можно использовать для определения вектора, способного хранить наборы любых типов. Следовательно, тип объекта iVec определяет выражение vector <int>. Более того, выражение vector <int> является не просто именем типа данных, это — имя класса. Аналогичные классы существуют и для других базовых типов данных. Таким образом, приведенное выше объявление создает объект iVec класса vector <int>. При объявлении объекта iVec используется конструктор по умолчанию, который создает пустой вектор, то есть вектор с нулевой длиной.

Точно так же, как и в массиве, элементы вектора нумеруются с 0. И для обращения к ним можно записать имя вектора и индекс в квадратных скобках. Например, iVec[1]. Однако это выражение не может использоваться для инициализации вектора (т.е. для присвоения начального значения), а только для изменения значения уже существующего элемента или для какого-либо другого использования этого значения, например, для вывода.

Для инициализации вектора можно использовать конструктор или последовательно добавлять в вектор элементы с помощью функции-члена push_back. Рассмотрим эти способы подробнее.

В классе vector определено несколько конструкторов, которые можно использовать при определении и инициализации объектов вектора:

```
vector <t> v1; //пустой вектор v1, который будет содержать объекты типа t
vector <t> v1(v2); //вектор v1 – копия вектора v2
vector <t> v1(n, i); //вектор v1 содержит n элементов типа t со значением i
vector <t> v1(n); //вектор v1 содержит n элементов типа t, значение которым
                  //присваивается в зависимости от типа t
```

Замечание. Если элементы вектора имеют базовый тип, например, int, то при использовании последнего конструктора элементы вектора примут значение 0. Если элементы

являются объектами класса, и для них определены собственные конструкторы, то для инициализации используется стандартный конструктор класса.

Главным преимуществом вектора является возможность динамически изменять свой размер по мере добавления в него элементов. Хотя количество элементов вектора можно задать заранее, как правило, удобнее создать пустой вектор и добавлять в него элементы по мере надобности. По мере добавления элементов размер вектора будет увеличиваться.

Элементы добавляются в вектор с помощью функции-члена push_back в определенном порядке: сначала в позицию 0, затем в позицию 1, позицию 2 и т. д. Таким образом, добавление элемента производится фактически в конец вектора.

Например, добавить элемент со значением i в конец вектора iVec можно следующим образом:

```
iVec.push_back(i);
```

Чтобы добавить в вектор элементы со значением 1,2,3,4,5 можно использовать следующий цикл:

```
for (int i=1;i<=5;i++)
  iVec.push_back(i);
```

Вывести элементы полученного вектора на экран можно, используя обращение по индексу, как в массиве:

```
for (unsigned int i=0;i<5;i++)
  cout <<iVec[i]<<endl;
```

Количество элементов вектора называется его *размером.* Для того чтобы узнать размер вектора, можно воспользоваться функцией-членом size(). Тогда вывод на экран всех элементов вектора iVec можно выполнить следующим образом:

```
for (unsigned int i=0;i<iVec.size();i++)
  cout <<iVec[i]<<endl;
```

Следует отметить, что функция size возвращает значение типа unsigned int, и тип индекса, с помощью которого мы обращаемся к элементам вектора тоже unsigned int. Поэтому переменная i объявлена типом unsigned int, а не int.

Замечание. Если вы объявите переменную i типом int, то компилятор выдаст предупреждение о том, что будет выполняться автоматическое преобразование типов.

Для вектора, точно так же, как и для строк можно использовать тип size_type. Поэтому вывод элементов вектора на экран можно организовать с помощью следующего цикла:

```
for (vector <int>::size_type i=0;i<iVec.size();i++)
  cout <<iVec[i]<<endl;
```

Кроме размера, для вектора существует еще такое понятие как емкость. В любой момент времени вектор имеет определенную емкость, то есть, определено количество элементов, для которых на этот момент выделена память. Емкость вектора можно узнать с помощью функции-члена capacity().

Замечание. Не путайте емкость вектора и его размер! Размер определяет количество элементов в заполненной части вектора, а емкость — это количество элементов, для которых выделена память. Емкость обычно больше размера, а не наоборот.

Когда емкость вектора исчерпана, она автоматически увеличивается. Насколько — зависит от конкретной реализации, но всегда на большую величину,

чем нужно в настоящий момент (как правило, она удваивается). Поскольку увеличение емкости вектора является достаточно сложной задачей, выделение памяти большими блоками эффективнее, чем маленькими.

Проиллюстрируем вышесказанное следующим примером:

```
#include "vector"
#include "iostream"
using namespace std;
int main ()
{
  vector <int> iVec;
  cout<<"Vector empty"<<endl;
  cout <<"size:"<<iVec.size()<<endl;
  cout <<"capacity:"<<iVec.capacity()<<endl;
  for (int i=1;i<=5;i++)
    iVec.push_back(i);
  cout<<"Vector is (1,2,3,4,5)"<<endl;
  cout <<"size:"<<iVec.size()<<endl;
  cout <<"capacity:"<<iVec.capacity()<<endl;
  for (int i=6;i<=10;i++)
    iVec.push_back(i);
  cout<<"Vector is (1,2,3,4,5,6,7,8,9,10)"<<endl;
  cout <<"size:"<<iVec.size()<<endl;
  cout <<"capacity:"<<iVec.capacity()<<endl;
  return 0;
}
```

После выполнения данной программы на экране будет выведено:

```
Vector empty
size: 0
capacity: 0
Vector is (1,2,3,4,5)
size: 5
capacity: 6
Vector is (1,2,3,4,5,6,7,8,9,10)
size: 10
capacity: 13
```

Проверить, является ли вектор пустым, можно с помощью функции-члена empty(). Она возвращает значение true, если вектор пуст и false в противном случае.

Сравнивать вектора можно с помощью операций ==, !=, <,>,<=,>=. Эти операции имеют обычное назначение. Так, например, операция == возвращает значение true, если операнды-вектора равны (имеют одинаковое количество одинаковых элементов, которые идут в одинаковом порядке) и false в противном случае.

Замечание. Самостоятельно проверьте работу остальных операций.

Вставлять элементы в вектор можно с помощью функции insert, а удалять – с помощью erase. Рассмотрим их подробнее.

Замечание. Следует отметить, что не следует злоупотреблять данными функциями, т.к. при каждой вставке или удалении будет изменяться структура вектора – он будет сжиматься или разжиматься, что будет снижать эффективность выполнения программы.

Функция insert(iter, value) вставляет элемент со значением value в позицию, которую указывает итератор iter. Так, например, для заданного вектор iVec, элементами которого являются целые числа, вставить элемент 10 во вторую позицию можно следующим образом:

```
iVec.insert(iVec.begin()+2, 10);
```

Замечание. Функция begin() возвращает итератор, которой позволяет обратиться к первому элементу вектора. Более подробно итераторы будет рассмотрены в следующем разделе.

Функция erase(iter) удаляет элемент из позиции, на которую указывает итератор iter. Например, удалить второй элемент из вектора можно следующим образом:

```
iVec.erase(iVec.begin()+2);
```

Метод erase(iter1,iter2) удаляет элементы из диапазона iter1, iter2, где iter1, iter2 – итераторы.

7.2. Итераторы

Кроме индексирования, для доступа к элементам вектора библиотека предоставляет еще один способ – итератор. Свой собственный итератор определен в каждом из классов контейнеров, в том числе, и в классе vector. Для нашего примера – вектора из целых чисел – итератор будет определяться следующим образом:

```
vector <int>::iterator iter;
```

В данном примере мы определили переменную iter типа iterator для класса vector <int>.

Замечание. Напомним, что тип с именем iterator определен в каждом библиотечном классе-контейнере.

В классе каждого контейнера определены две функции begin() и end(), которые возвращают итератор. Итератор, возвращаемый функцией begin() позволяет обратиться к первому элементу контейнера, если он есть. Обращение iVec.begin() в случае если вектор не пуст, эквивалентно обращению iVec[0].

Итератор, возвращаемый функцией end(), указывает на элемент, следующий за последним в векторе. Иногда говорят, что он указывает на конец вектора, но на самом деле, если им воспользоваться, то произойдет выход за границу вектора. Этот итератор используется как граница при переборе элементов вектора. В случае если вектор пуст, функции begin() и end() возвращают одинаковый итератор.

Операции с итераторами

Операции с переменными типа iterator позволяют получить доступ к элементу, на который указывает итератор, а также переместить итератор с одного элемента на другой.

Для доступа к элементу, на который указывает итератор, используется операция обращения к значению *. Т.е. *iter возвращает элемент, на который указывает итератор iter.

Чтобы переместить итератор на следующий элемент в контейнере, используется оператор инкремента ++. Т.е. если iter указывал на первый элемент

вектора, то после выполнения ++iter он будет указывать на второй элемент. Аналогично с итераторами используется оператор декремента --.

Итераторы векторов поддерживают также другие арифметические операции. Например, к итератору можно прибавить (вычесть) целочисленное значение:

iter+n; или iter-n;

В результате получается новый итератор, который указывает на n элементов вперед (или назад) от исходного значения итератора iter. Результат сложения должен указывать на элемент вектора или одну из его границ. Типом полученного значения является обычно size_type или difference_type. Последний также определен в классе вектора и является знаковым.

Кроме того, можно сложить или вычесть два итератора, относящиеся к одному вектору. Результатом будет знаковое целочисленное значение типа difference_type.

Проиллюстрируем вышесказанное на примере вывода элементов вектора iVec:

```
for (vector <int>::iterator iter=iVec.begin();iter<iVec.end();iter++)
    cout <<*iter<<endl;
```

Замечание. Этот фрагмент безошибочно сработает и с пустым вектором. В этом случае итератор, который возвращает функция begin() совпадает с итератором, который возвращает функция end(), и после первой проверки условия цикл выполняться не будет.

7.3. Алгоритмы STL

В библиотечных контейнерах определен некоторый набор функций, предназначенный для их обработки. В дополнение к нему STL предоставляет набор алгоритмов, которые не зависят от конкретного типа контейнера и выполняют такие действия как поиск, замену, удаление элементов в контейнерах, сортировку элементов контейнеров и многие другие. Аргументами этих алгоритмов служат итераторы. Для обеспечения доступа к алгоритмам STL нужно подключить заголовочный файл "algorithm".

Замечание. Алгоритмы STL можно применять и к массивам, если в качестве итераторов использовать указатели на элементы массива.

Рассмотрим наиболее интересные алгоритмы для работы с векторами. Если не будет оговорено другого, то в качестве параметров алгоритма мы будем использовать итераторы first и last, обозначающие соответственно начало и конец диапазона, в котором алгоритмом производятся соответствующие действия.

Алгоритм find(first, last, value) ищет в заданном диапазоне первый элемент, значение которого равно value. Возвращает итератор на найденный элемент.

Алгоритм count(first, last, value) подсчитывает, сколько раз в заданном диапазоне встречается элемент со значением value. Возвращает найденное количество.

Алгоритм sort(first,last) – сортирует объекты в заданном диапазоне.

Алгоритм replace(first, last, old, new) – в заданном диапазоне заменяет все объекты равные old, объектами, равными new.

Алгоритм remove(first, last, value) удаляет из заданного диапазона все объекты, равные value. Реализовывается это следующим образом. Элементы, которые надо удалить, перемещаются в конец контейнера, и алгоритм возвращает итератор, который указывает на начало последовательности удаляемых элементов.

Алгоритм min_element(first, last, value) возвращает итератор, указывающий на наименьший элемент в заданном диапазоне.

Алгоритм max_element(first, last, value) возвращает итератор, указывающий на наибольший элемент в заданном диапазоне.

Алгоритм iter_swap(iter1,iter2) меняет местами объект, на который указывает итератор iter1 с объектом, на который указывает итератор iter2.

Для иллюстрации рассмотрим следующие примеры.

1. Дана последовательность из n элементов. Поменять местами максимальный и минимальный элемент:

```
#include "iostream"
#include "algorithm"
#include "vector"
using namespace std;
int main()
{
 vector <int> iVec;
 int x,n;
 cout<<"n="; cin>>n;
 //В цикле формируем вектор из n элементов, значения которых вводятся с
 //клавиатуры. Добавление элементов производится в конец вектора с
 //помощью функции push_back
 for (int i=0;i<n;i++)
  {
   cout<<"Введите элемент с номером "<<i<<endl;
   cin>>x;
   iVec.push_back(x);
  }
 //с помощью алгоритма min_element находим итератор iterMin, который
 //указывает  на минимальный элемент в векторе
 vector <int>::iterator iterMin=min_element(iVec.begin(),iVec.end());
 //выводим минимальный элемент на экран
 cout<<"min="<<*iterMin<<endl;
 //с помощью алгоритма max_element находим итератор iterMin, который
 //указывает на максимальный элемент
 vector <int>::iterator iterMax=max_element(iVec.begin(),iVec.end());
 //выводим максимальный элемент
 cout<<"max="<<*iterMax<<endl;
 //с помощью алгоритма iter_swap(iterMin,iterMax) меняем местами элемент, на
 //который указывает iterMin и элемент, на который указывает iterMax
 iter_swap(iterMin,iterMax);
 //выводим на экран измененный вектор
 for (vector <int>::iterator iter=iVec.begin();iter<iVec.end();iter++)
   cout <<*iter<<endl;
 return 0;
}
```

2. Дана последовательность из n элементов. Удалить все элементы, равные 0.

```
#include "iostream"
#include "algorithm"
#include "vector"
using namespace std;
int main()
{
 vector <int> iVec;
 int x,n;
 cout<<"n="; cin>>n;
 for (int i=0;i<n;i++)
  {
   cout<<"vvedite "<<i<<" el vectora"<<endl;
   cin>>x;
   iVec.push_back(x);
  }
 //элементы, равные нулю, перемещаются в конец вектора и располагаются,
 //начиная с позиции, на которую указывает iEnd
 vector <int>::iterator iEnd=remove(iVec.begin(),iVec.end(),0);
 //тогда вывести нам нужно элементы с начала вектора до позиции iEnd
 for (vector <int>::iterator iter=iVec.begin();iter<iEnd;iter++)
   cout <<*iter<<endl;
 return 0;
}
```

Чтобы физически удалить нулевые элементы из вектора, можно после функции remove использовать функцию erase. Тогда конец программы будет выглядеть следующим образом:

```
vector <int>::iterator iEnd=remove(iVec.begin(),iVec.end(),0);
iVec.erase(iEnd,iVec.end());
for (vector <int>::iterator iter=iVec.begin();iter<iVec.end();iter++)
  cout <<*iter<<endl;
```

Добавление _if к названию алгоритма

Некоторые алгоритмы имеют версии с окончанием _if (например, find_if, count_if, replace_if, remove_if). Для версий функций с окончанием _if требуется дополнительный параметр, который называется предикатом, и который в свою очередь также является функцией.

Например, для того чтобы найти в контейнере iVec элемент со значением 5, мы будем использовать следующий вызов функции find:

```
vector <int>::iterator iter=find(iVec.begin(),iVec.end(),5);
```

Рассмотрим теперь как найти в контейнере iVec первый положительный элемент. Будем использовать для этого функцию find_if, которая в качестве одного из аргументов использует предикат. Предикат – это функция, которая возвращает значение true, если условие выполняется и false в противном случае. В нашем случае условием будет «аргумент больше нуля». Тогда функцию можно задать следующим образом:

```
bool Pred(int x)
{
  return (x<0);
}
```

И обращение к find_if для решения нашей задачи можно записать следующим образом:

```
vector <int>::iterator it=find_if(iVec.begin(),iVec.end(),Pred);
```

Пример. В заданном наборе целых чисел все отрицательные элементы заменить на 0, используя алгоритм replace_if.

1 вариант. Работа с массивом.

Замечание. Если мы описали одномерный массив a из n элементов, то обратиться к его элементам мы можем через индекс или через указатель. Например, чтобы обратиться к нулевому элементу массива можно написать a[0] или *a. Чтобы обратиться к элементу с номером i – a[i] или *(a+i) соответственно. Чтобы обратиться к последнему элементу нужно написать a[n-1] или *(a+n-1). Таким образом, указатель на последний элемент в массиве будет задаваться выражением a+n-1.

```
#include "iostream"
#include "algorithm"
using namespace std;
//функция возвращает значение true, если аргумент отрицательный, в
//противном случае возвращает значение  false
bool isNeg(int x)
{
  return (x<0);
}
int main()
{
 int n;
 int a[10];
 cout<<"n="; cin>>n
 for (int i=0;i<n; i++)        //вводим  элементы массива
  {
    cout<<"a["<<i<<"]="; cin>>a[i];
  }
 //все элементы в диапазоне от a ( указатель на первый элемент массива), до
 //a+n-1 (указатель на последний элемент массива), для которых выполняется
 //предикат isNeg (т.е.элементы являются отрицательными), заменяем 0
 replace_if(a,a+n-1,isNeg,0);
  for (int i=0;i<n; ++i)          //выводим измененный массив
    cout<<"a["<<i<<"]="<<a[i]<<endl;
return 0;
}
```

2. вариант. Работа с вектором.

Замечание. В данном случае при обращении к алгоритму replace_if первым и вторым аргументом будут итераторы, указывающие на первый и последний элемент вектора соответственно.

```
#include "iostream"
#include "algorithm"
#include "vector"
using namespace std;
bool isNeg(int x)
{ return (x<0); }
int main()
{
 vector <int> iVec;
 int x,n;
 cout<<"n="; cin>>n;
 //формируем вектор из n элементов, значения которых вводятся с клавитуры
 for (int i=0;i<n;i++)
  {
   cout<<"Введите элемент вектора с номером "<<i<<endl;
   cin>>x;
   iVec.push_back(x);
  }
 //все элементы вектора в диапазоне от iVec.begin() – итератор, указывающий
 //на первый элемент вектора, до iVec.end() – итератор, указывающий на конец
 //вектора, для которых выполняется предикат isNeg (т.е. элементы вектора
 //являются отрицательными) заменяем 0
 replace_if(iVec.begin(),iVec.end(),isNeg,0);
 //выводим измененный вектор
 for (vector <int>::iterator iter=iVec.begin();iter<iVec.end();iter++)
   cout <<*iter<<endl;
 return 0;
}
```

Обобщенные числовые алгоритмы

В STL существуют так называемые обобщенные числовые алгоритмы, для работы с которыми необходимо подключить заголовочный файл "numeric". Рассмотрим для примера один из них.

Алгоритм accumulate(first,last,init) считает сумму элементов из заданного диапазона, где init – это начальное значение этой суммы. Алгоритм возвращает результат суммирования. Начальное значение указывает функции тип суммируемых элементов и оно должно совпадать с типом элементов контейнера или допускать преобразование в тип элементов контейнера. Тогда подсчитать сумму элементов вектора iVec, состоящего из целых чисел, можно следующим образом:

```
int sum=accumulate(iVec.begin(),iVec.end(),0);
```

7.4. Упражнения

I. Дана последовательность целых чисел

1. Заменить все положительные элементы на x.
2. Заменить все четные элементы на x.
3. Заменить все элементы, попадающие в интервал [a, b], нулем.
4. Заменить все элементы, значения которых равны x на y.

5. Заменить все двухзначные числа на x.
6. Заменить все простые числа на x.
7. Подсчитать количество четных элементов.
8. Подсчитать количество элементов, кратных 9.
9. Подсчитать количество элементов, не попадающих в заданный интервал.
10. Определить, является ли сумма элементов двухзначным числом.
11. Определить, является ли сумма элементов простым числом.
12. Подсчитать количество максимальных элементов.
13. Заменить все максимальные элементы нулями.
14. Заменить все минимальные элементы данным числом x.
15. Поменять местами максимальный элемент и первый.
16. Подсчитать сумму элементов, расположенных между максимальным и минимальным элементами (минимальный и максимальный элементы в массиве единственные). Если максимальный элемент встречается позже минимального, то выдать сообщение об этом.
17. Подсчитать сумму элементов, расположенных между минимальным и максимальным элементами (минимальный и максимальный элементы в массиве единственные). Если минимальный элемент встречается позже максимального, то выдать сообщение об этом.
18. Поменять местами первый минимальный и последний элементы.
19. Выполнить перестановку элементов по правилу: первый с последним, второй с предпоследним и т.д.
20. Выполнить перестановку элементов по правилу: первый со вторым, третий с четвертым и т.д.

II. Дана последовательность целых чисел.

1. Удалить из массива все четные числа.
2. Удалить из массива все максимальные элементы.
3. Удалить из массива все числа, значения которых попадают в данный интервал.
4. Удалить из массива все элементы, последняя цифра которых равна x.
5. Удалить из массива элементы с номера k1 по номер k2.
6. Удалить из массива каждый k-тый по счету элемент.
7. Удалить из массива последний минимальный элемент.
8. Вставить новый элемент перед первым отрицательным элементом.
9. Вставить новый элемент после последнего положительного.
10. Вставить новый элемент перед всеми четными элементами.
11. Вставить новый элемент после всех элементов, которые заканчиваются на заданную цифру.
12. Вставить новый элемент после всех элементов, кратных своему номеру.
13. Перед каждым k-тым по счету элементом вставить 0.
14. Удвоить каждый k-тый элемент.
15. Удвоить каждый отрицательный элемент.
16. После каждого k-того по счету элемента вставить новое значение.
17. Удалить из массива все элементы, в записи которых все цифры различны.
18. Удалить из массива повторяющиеся элементы, оставив только их первые вхождения.
19. Вставить новый элемент после всех максимальных.

20. Вставить новый элемент перед всеми элементами, в записи которых есть данная цифра.

8. ИСКЛЮЧЕНИЯ

Язык C++, как и многие другие объектно-ориентированные языки, реагирует на ошибки и ненормальные ситуации с помощью механизма обработки *исключений.* Исключение - это объект, генерирующий информацию о «необычном программном происшествии». При этом механизм обработки исключений в C++ не поддерживает обработку асинхронных событий, таких как ошибки оборудования или обработку прерываний, например, нажатие комбинаций клавиш на клавиатуре. Механизм обработки исключений предназначен только для событий, которые возникают в результате работы самой программы. Для программиста важно понимать разницу между ошибкой в программе и исключительной ситуаций.

Ошибка в программе допускается программистом при ее разработке. Например, вместо операции сравнения (==) используется операция присваивания (=). Программист должен исправить подобные ошибки на этапе отладки программы. Использование механизма обработки исключений не является защитой от ошибок данного вида.

Даже если программист исправил все свои ошибки в программе, он может столкнуться с непредсказуемыми и неотвратимыми проблемами - *исключительными ситуациями.* Например, нехваткой доступной памяти или попыткой открыть несуществующий файл. Исключительные ситуации программист предвидеть не может, но он может отреагировать на них так, чтобы они не привели к краху программы. Для обработки исключительных ситуаций в C++ используется специальный механизм обработки исключений.

8.1. Механизм обработки исключений

Механизм обработки исключений разделяет вычислительный процесс на две части – обнаружение исключительной ситуации и ее обработка. Рассмотрим данный механизм более подробно.

Обработка исключения начинается с появления ошибки. Функция, в которой она возникла, генерирует исключение с помощью оператора throw с параметром, определяющим тип исключения. Параметр может быть константой, переменной или объектом и используется для передачи информации об исключении его обработчику. Например:

```
throw 1;   //параметр – целочисленная константа
throw "Ошибка: деление на ноль"; //параметр – строковая константа
throw k; //параметр – переменная
```

Сгенерированное исключение нужно перехватить и обработать. Перехват исключения и проверка возникновения исключения выполняется с помощью оператора try, с которым неразрывно связан один или несколько блоков обработки исключений – catch. Оператор try объявляет в любом месте программы контролируемый блок, который имеет следующий вид:

```
try    // контролируемый блок
{ ...}
```

Программные инструкции, которые нужно проконтролировать на предмет исключений, помещаются в контролируемый блок. Контролируемый блок, помимо функций контроля, обладает функциями обычного блока: все идентификаторы объявленные внутри него, являются локальными в этом блоке и не видны вне его.

За блоком try следует один или несколько блоков catch – блоков обработки исключений. Формат записи блока catch следующий:

```
catch (спецификация_параметра_исключения)      // блок обработки
{ ...}
```

Спецификация параметра исключения может иметь три формы:

1. (тип имя)
2. (тип)
3. (...)

Первый вариант спецификации означает, что объект-исключение передается в блок обработки для использования, например, для вывода информации об аварийной ошибке. При этом объект-исключение может передаваться в блок catch по значению, по ссылке или по указателю. Например:

```
catch (int exeption) { ...}          //по значению
catch (int & exeption) { ...         //по ссылке
catch (const int & exeption) { ...}  //по константной ссылке
catch (int *exeption) { ...}         //по указателю
```

Вторая форма спецификации предназначена для обработки исключения конкретного типа, но без передачи объекта-исключения в блок catch. Например:

```
catch (int) { ...}          //обработка исключений целого типа
catch (double) { ...}       //обработка исключений вещественного типа
```

Третий вариант спецификации используется для обработки исключений всех типов.

Работа конструкции try-catch напоминает работу оператора scwith. Если в блоке try сгенерировано исключение, то начинается сравнение типа сгенерированного исключения с типами параметров блоков catch. Выполняется тот блок catch, тип которого совпал с типом исключения. Если блок catch с заданным типом не найден, то выполняется блок catch с многоточием. Если же такого блока нет, то он ищется в вызывающей функции, причем поиск продолжается до функции main. Если же и там не обнаруживается нужного блока catch, то вызывается стандартная функция завершения terminate() (см. п.п.15.5.1 в [1]), которая вызывает функцию abort(). Таким образом, очень важен порядок записей блоков catch – если в качестве первого блока указать блок с параметром-многоточием, то такой обработчик будет обрабатывать все типы исключений и до остальных блоков catch просто не дойдет. Поэтому нужно усвоить следующее правило – блок catch с параметром-многоточием должен быть последним.

Следует отметить, что выход из блока catch выполняется одним из способов:

1. Выполняются все операторы блока catch, после чего управление передается оператору, расположенному после конструкции try-catch.
2. В блоке catch выполняется оператор goto, при этом разрешается переходить на любой оператор вне конструкции try-catch (внутрь контролируемого блока или в другой блок catch переход запрещен).
3. В блоке catch выполняется оператор return, после чего происходит нормальный выход из функции.
4. В блоке catch повторно генерируется другое исключение.

Рассмотрим следующий пример:

```
#include "iostream"
using namespace std;
int main()
{
  try
  {
    throw "сгенерировано исключение";
    cout<<"действия после генерации исключения"<<endl;
  }
  catch (int)
  {
    cout<<"целочисленная ошибка"<<endl;
  }
  catch (char *s)
  {
    cout<<s<<endl;
  }
  catch (...)
  {
    cout<<"возникла какая-то ошибка"<<endl;
  }
cout<<"конец программы"<<endl;   //1
}
```

Когда генерируется исключение, выполнение программы останавливается и управление передается блоку *catch* соответствующего типа. Этот блок *никогда* не возвращает управление в то место программы, где возникло исключение. Поэтому команды из блока *try*, расположенные ниже строки, в которой было сгенерировано исключение, никогда не будут выполнены. В нашем примере сгенерировано исключение строкового типа, поэтому управление будет передано блоку *catch (char *s)* и на экран будет выведено сообщение "сгенерировано исключение". После чего управление будет передано строке с номером 1 и на экран будет выведено сообщение "конец программы".

8.2. Применение исключений на практике

Одно из основных достоинств обработки исключений состоит в том, что она позволяет программе отреагировать на возможную ошибку при вычислениях, а затем продолжить выполнение без прерывания работы программы. Например, нам

нужно построить таблицу значений для функции вида $y(x)=\frac{100}{x^2-1}$ на отрезке [a, b] с шагом h. Тогда программа может выглядеть следующим образом:

```
#include "iostream"
#include "iomanip"
using namespace std;
int main()
{
  double a,b,h, x, y;
  cout<<"a= "; cin>>a;
  cout<<"b= "; cin>>b;
  cout<<"h= "; cin>>h;
  cout<<left<<setprecision(5);
  cout<<setw(10)<<'x'<<setw(10)<<'y'<<endl;
  for (x=a; x<=b; x+=h)
  {
      try
        {
          if (x==1 || x ==-1)throw "деление на ноль";
          y=100.0/(x*x-1);
          cout<<setw(10)<<x<<setw(10)<<y<<endl;
        }
      catch (char *s)
        {
          cout<<setw(10)<<x<<s<<endl;
        }
  }
  return 0;
}
```

входные данные	*выходные данные*	
a=-2	x	y
b=2	-2	33.33
h=1	-1	деление на ноль
	0	-100
	1	деление на ноль
	2	33.33

В некоторых случаях исключительная ситуация может возникнуть во вспомогательной функции. Информацию об этом можно передать в вызывающую функцию с помощью повторной генерации исключения. Например, построить таблицу значений для функции вида $y(x)=\frac{100}{x^2-1}$ на отрезке [a, b] с шагом h с использованием вспомогательной функцией. Тогда программа будет выглядеть следующим образом:

```
#include "iostream"
#include "iomanip"
using namespace std;
double f (double x) //вспомогательная функция
```

```
{
  try
    {
      if (x==1 || x ==-1)throw "деление на ноль"; //генерация исключения
    }
  catch (char *s)   //обработка исключения
    {
      //повторная генерация исключения, информация о котором передается в
      //вызывающую функцию
      throw s;
    }
   return 100.0/(x*x-1);
}
int main() //основная функция
{
  double a,b,h, x, y;
  cout<<"a= "; cin>>a;
  cout<<"b= "; cin>>b;
  cout<<"h= "; cin>>h;
  cout<<left<<setprecision(5);
  cout<<setw(10)<<'x'<<setw(10)<<'y'<<endl;
  for (x=a; x<=b; x+=h)
  {
      try //пытаемся выполнить вспомогательную функцию
       {
          y=f(x);
          cout<<setw(10)<<x<<setw(10)<<y<<endl;
       }
      //обработка исключения, переданного из вспомогательной функции
     catch (char *s)
     {
       cout<<setw(10)<<x<<s<<endl;
     }
  }
  return 0;
}
```

Замечания.

1. Результат выполнения данной программы будет соответствовать предыдущему примеру.
2. Чтобы более подробно познакомиться с механизмом обработки исключительных ситуаций, рекомендуем воспользоваться дополнительной литературой.

8.3. Упражнения

Постройте таблицу значений функции y=f(x) для $x \in [a, b]$ с шагом h. Если в некоторой точке *x* функция не определена, то выведите на экран сообщение об этом.

Замечание. При решении данной задачи использовать вспомогательную функцию f(x), которая вычисляет значение y, а также проводить обработку возможных исключений.

1. $y = \frac{1}{(1+x)^2}$;

2 $y = \frac{1}{x^2-1}$;

3. $y = \sqrt{x^2-1}$

4. $y = \sqrt{5-x^3}$;

5. $y = \ln(x-1)$;

6. $y = \ln(4-x^2)$;

7. $y = \frac{x}{\sqrt{2x-1}}$;

8. $y = \frac{3x+4}{\sqrt{x^2+2x+1}}$;

9. $y = \frac{1}{x-1} + \frac{2}{1-4x}$;

10. $y = \ln|x-2|$;

11. $y = \ln\frac{x}{x-2}$;

12. $y = \ln(x^4-1)\ln(1+x)$;

13. $y = \frac{\ln(x-2)}{\sqrt{5x+1}}$;

14. $y = \frac{\sqrt{x^2-2x+1}}{\ln(4-2x)}$;

15. $y = \ln|3x|\sqrt{2x^5-1}$;

16. $y = \frac{3}{|x^3+8|}$;

17. $y = \frac{x+4}{x^2-2} + \sqrt{x^3-1}$;

18. $y = \sqrt{x^2-1} - \sqrt{x^2-5}$;

19. $y = \frac{\sqrt{x^3-1}}{\sqrt{x^2-1}}$;

20. $y = \frac{1}{x+7} + \ln(1-|x|)$.

ЛИТЕРАТУРА

1. International Standart ISO/IEC 14882:2003(E), Programming languages – C++.
2. *Альфред В. Ахо, Джон Э. Хопкрофт, Джеффри Д. Ульман.* Структуры данных и алгоритмы.: – М.: Издательский дом «Вильямс», 2000.
3. *Вирт Н.* Алгоритмы и структуры данных: Пер. с англ. - М.: Мир, 1989.
4. *Керниган Б., Ритчи Д.* Язык программирования Си. \ Пер. с англ., 3-е изд., испр. – СПб.: «Невский диалект», 2001.
5. *Климова Л.М.* Си++. Практическое программирование. Решение типовых задач. М.:КУДИЦ-ОБРАЗ, 2001.
6. *Кнут Д.* Искусство программирования для ЭВМ. Т.1. Основные алгоритмы. – М: Мир, 1976.
7. *Кормен Т., Лейзерсон Ч., Ривест Р.* Алгоритмы: построение и анализ. – М: МЦНМО, 2001.
8. *Лаптев В.В.* С++ Объектно-ориентированное программирование. . – СПб: Питер, 2008.
9. *Лафоре Р.* Объектно-ориентированное программирование в С++. Классика Computer Science. 4-е изд. – СПб.: Питер, 2007.
10. *Липпман С.Б., Лажойе Ж.* Язык программирования С++. Вводный курс. 4-е изд-е. Изд. дом "Вильямс", 2007 г..
11. *Павловская Т.А.* С/С++. Программирование на языке высокого уровня. – СПб: Питер, 2006.
12. *Страуструп Бьерн.* Язык программирования С++. \ Пер. с англ., 3-е изд. – СПб.; М.: «Невский диалект» - «Издательство БИНОМ», 1999.
13. *Шилдт Г.* Самоучитель С++: Пер. с англ. — 3-е изд. — СПб.: БХВ-Петербург, 2005.

Содержание