А.М. НИГМЕДЗЯНОВА, Ф.Т. ГАТАУЛЛИН

РЕШЕНИЕ ЗАДАЧ ПРОГРАММИРОВАНИЯ ДЛЯ ПОДГОТОВКИ К ЕГЭ ПО ИНФОРМАТИКЕ НА ЯЗЫКЕ С/С++

Учебно-методическое пособие

Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное образовательное
учреждение высшего образования
«**КАЗАНСКИЙ (ПРИВОЛЖСКИЙ) ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ**»

А.М. НИГМЕДЗЯНОВА, Ф.Т. ГАТАУЛЛИН

«РЕШЕНИЕ ЗАДАЧ ПРОГРАММИРОВАНИЯ ДЛЯ ПОДГОТОВКИ К ЕГЭ ПО ИНФОРМАТИКЕ НА ЯЗЫКЕ C/C++»

Учебно-методическое пособие

Казань 2019

УДК 004.42
ББК 32.973-018.1(07)

Печатается по решению учебно - методической комиссии Института математики и механики им. Н.И.Лобачевского.

А.М. Нигмедзянова, Ф.Т. Гатауллин

Решение задач программирования для подготовки к ЕГЭ по информатике на языке C/C++ \Нигмедзянова А.М., Гатауллин Ф.Т. - Казань: Казанский федеральный университет, 2019. -97 с.

Данное учебно-методическое пособие содержит теоретический и практический материал по основам программирования на языке C/C++. Содержит разбор конкретных задач из контрольно-измерительного материала ЕГЭ, а так же задания для самостоятельного решения. Пособие предназначено для обучающихся старших классов общеобразовательных школ, лицеев и колледжей и учителей информатики, а так же для студентов направления «Педагогическое образование» по профилю «Информатика».

Содержание

Введение

Изучение программирования занимает важное место в развитии мышления школьников, в выработке многих приемов умственной деятельности. Изучая основы программирования, учащиеся приобретают алгоритмическое мышление, познают азы профессии программиста и получают возможность качественнее подготовиться к сдаче ЕГЭ по информатике.

Данное учебно-методическое пособие предназначено для школьников 9-11 класса, а так же для учителей информатики. В этом пособии рассмотрена теория программирования на языке C++, необходимая для решения задач ЕГЭ, а также приведены и решены примеры из контрольно-измерительного материала ЕГЭ 2018 и предложены задания для самостоятельного решения различных уровней сложности. Пособие дает выпускникам не только необходимые знания для успешной сдачи ЕГЭ, но и позволяет развивать логическое мышление, расширяет границы формирующегося представления школьников об области информационных технологий, повышает информационную грамотность, а также помогает определиться с выбором профессионального пути.

1 ГЛАВА 1. Теоретическая часть

1.1 Простейшие программы. Вычисления. Стандартные функции

1.1.1 Алгоритм и его свойства

Алгоритмом называется порядок действий, которые исполнитель за определенное время должен выполнить для достижения цели.

Здесь исполнитель - это устройство или одушевленное существо (человек), способное понять и выполнить команды, составляющие алгоритм.

Человек как исполнитель часто действует неформально, по-своему понимая команды. Несмотря на это, ему тоже часто приходится действовать по тому или иному алгоритму. Например, нахождение корней квадратного уравнения производится по алгоритму: Сначало надо выяснить дискриминант квадратного уравнения, если дискриминант больше 0, то квадратное уравнение имеет два корня, если дискриминант равен 0, то имеется только одно решение, и в случае если дискриминант меньше 0, тогда корень вычислить нельзя. Когда нашли дискриминант, подставляем его в формулу нахождения корней и находим корни уравнения. При сборе сумки в школу вы тоже действуете по определенному алгоритму [9].

Свойства алгоритма:

— **Дискретность** - алгоритм состоит из отдельных команд (шагов), каждая из которых выполняется в ограниченное время.

— **Детерминированность (определенность)** - при каждом запуске алгоритма с одними и теми же исходными данными должен быть получен один и тот же результат.

— **Понятность** - алгоритм содержит только команды, входящие в систему команд исполнителя, для которого он предназначен.

— **Конечность (результативность)** - для корректного набора данных алгоритм должен завершаться через конечное время с вполне определенным результатом (результатом может быть сообщение о том, что задача не имеет решений).

— **Корректность** - для допустимых исходных данных алгоритм должен приводить к правильному результату.

— **Массовость** - алгоритм, как правило, предназначен для решения множества однотипных задач с различными исходными данными[9].

Эти свойства не равноправны. Дискретность, детерминированность и понятность - фундаментальные свойства алгоритма, то есть ими обладают все алгоритмы для формальных исполнителей. Остальные свойства можно

рассматривать как требования к «правильному» алгоритму.

Иными словами, алгоритм получает на вход некоторый дискретный входной объект (например, набор чисел или слово) и обрабатывает входной объект по шагам (дискретно), строя промежуточные дискретные объекты. Этот процесс может закончиться или не закончиться. Если процесс выполнения алгоритма заканчивается, то объект, полученный на последнем шаге работы, является результатом работы алгоритма при данном входе. Если процесс выполнения не заканчивается, говорят, что алгоритм зациклился. В этом случае результат его работы не определен[9].

Способы записи алгоритмов

Алгоритмы можно записывать разными способами:

— **на естественном языке**, обычно такой способ применяют, записывая основные идеи алгоритма на начальном этапе;

— на **псевдокоде**, так называется смешанная запись, в которой используется естественный язык и операторы какого-либо языка программирования; в сравнении с предыдущим вариантом такая запись гораздо более строгая;

— в виде **блок-схемы** (графическая запись);

— в виде **программы** на каком-либо языке программирования.

Из всех перечисленных здесь способов мы будем использовать два: псевдокод и запись алгоритма в виде программы на языке программирования C++[9].

1.1.2 Простейшие программы

Пустая программа - это программа, которая ничего не делает, но удовлетворяет требованиям выбранного языка программирования. Она полезна, прежде всего, для того, чтобы понять общую структуру программы.

```
main()
{
// это основная программа
//здесь записывают операторы//
}
```

Основная программа всегда называется именем **main** (от англ. *main* - основной, главный). Пустые скобки означают, что она не принимает никаких дополнительных данных (параметров) от пользователя.

Тело (основная часть) программы ограничено фигурными скобками. Между ними записывают *операторы* - команды языка программирования. Все, что следует после символов // до конца строки, это *комментарии* - пояснения, которые не обрабатываются транслятором. Комментарий можно ограничивать парами символов /* и */, в этом случае комментарий может занимать несколько строк[9].

Вывод текста на экран

Напишем программу, которая выводит на экран такие строки:

$$2 + 2 =?$$

Ответ: 4

Перед заголовком основной программы добавились строки, начинающиеся с символов # **include** (от англ. *include* - включить). Это команды (инструкции, директивы) для препроцессора - программы, которая обрабатывает текст нашей программы до того, как за него «примется» транслятор, генерирующий машинные коды. С помощью команд # **include** в программу включаются *заголовочные файлы*, в которых описаны стандартные функции языка программирования и дополнительных библиотек[9].

Ввод и вывод в С++. Подключается заголовочный файл **iostream** (от англ. *input output stream - поток ввода и вывода*). *Поток вывода* (то есть, последовательность символов, выводимая на экран) называется **cout** (от англ. *console output* - вывод на консоль), а поток ввода (последовательность символов, вводимая с клавиатуры) имеет имя **cin** (от англ. *console input* - ввод с консоли). Справа от оператора << записывают данные, которые нужно вывести на экран, в нашей программ - это строки в кавычках. Можно записывать несколько операторов << в одной команде, например:

```
cout << \2+\<< \2=?\n\ << \Ответ: 4\;
```

Вместо символов **\n** для перехода на новую строку можно использовать специальную инструкцию (манипулятор) **endl**:

```
cout << \2+\ << \2=?\ << endl << \Ответ: 4\;
```

Обратите внимание, что любая команда заканчивается точкой с запятой.

Последняя инструкция **cin.get()** в программе на С++ обеспечивает задержку до нажатия на клавишу *Enter*. Дословно она означает «получить символы из входного потока и ничего с ними не делать».

Строчка

```
using namespace std;
```

говорит о том, что будет использоваться пространство имен **std**, в котором определена вся стандартная библиотека языка C++. Здесь пространство имен - это область действия объектов программы, в том числе переменных. В области **std** описаны стандартные потоки ввода и вывода с именами **cin** и **cout**.

```
#include <iostream> main()
{
std::cout<<"2+";
std::cout<<"2=?\n";
std::cout<<"Ответ:4";
std::cin.get();
}
```

1.1.3 Переменные

Напишем программу, которая выполняет сложение двух чисел:

1) запрашивает у пользователя два целых числа;

2) складывает их;

3) выводит результат сложения.

Программу на псевдокоде (смеси русского языка и языка С) можно записать так :

```
main()
{
// ввести два числа
// сложить их
// вывести результат
}
```

Компьютер не может выполнить псевдокод, потому что команд «ввести два числа» и ей подобных, которые записаны в комментариях, нет в его системе команд. Поэтому нам нужно «расшифровать» все такие команды через операторы языка программирования.

В отличие от предыдущих задач, данные нужно хранить в памяти. Для этого используют *переменные*.

Переменная - это величина, которая имеет имя, тип и значение. Значение переменной может измениться во время выполнения программы.

Переменная (как и любая ячейка памяти) может хранить только одно значение. При записи в нее нового значения «старое» стирается и его уже никак не восстановить.

Переменные в программе необходимо объявлять. При объявлении указывается тип переменной и ее имя (*идентификатор*). Значение переменной сразу после объявления не определенно: переменной выделяется некоторая область памяти, и там могло быть до этого записано любое число.

Вот так объявляются целочисленные переменные (в которых могут храниться только целые значения) с именами **a**, **b** и **c**:

```
int a, b, c;
```

Описание переменных начинается с ключевого слова, которое определяет тип переменных, а после него записывают список имен переменных. В данном случае указан тип **int** (от англ. *integer* - целочисленный)[9].

В именах переменных можно использовать латинские буквы (строчные и заглавные буквы различаются), цифры (но имя не может начинаться с цифры, иначе транслятору будет сложно различить, где начинается имя, а где - число) и знак подчеркивания «_».

Желательно давать переменным «говорящие» имена, чтобы можно было сразу понять, какую роль выполняет та или иная переменная.

Тип переменной нужен для того, чтобы

— определить область допустимых значений переменной;

— определить допустимые операции с переменной;

— определить, какой объем памяти нужно выделить переменной и в каком формате будут храниться данные.

— предотвратить случайные ошибки (опечатки в именах переменных).

После объявления переменной ее значение не определено (в ней записан «мусор» - неизвестное значение) и использовать ее нельзя. Если нужно, прямо при объявлении можно задать начальные значения переменных:

```
int a = 1, b, c = 123;
```

В этом примере переменной **b** не присваивается начальное значение, в ней содержится «мусор»[9].

Приведем полную программу сложения двух чисел:

```
#include <iostream>
using namespace std;
main()
{
int a, b, c;
cin>>a>>b;
c=a+b;
cout<<c;
cin.get(); cin.get();
}
```

Команда **cin.get()** повторяется два раза. Это сделано потому, что при первом вызове она прочитает все, что осталось во входном потоке после записи значения переменной **b** (хотя бы символ «новая строка»), и при однократном вызове программа не будет ждать нажатия на клавишу *Enter*.

Оператор, содержащий символ «=» - это **оператор присваивания**, с его помощью изменяют значение переменной. Он выполняется следующим образом: вычисляется выражение справа от символа «=», а затем результат записывается в переменную, записанную слева. Поэтому, например, оператор

```
i= i+1;
```

увеличивает значение переменной **i** на 1. Часто используется также сокращенная запись

```
i++;
```

которая обозначает то же самое. Аналогичная запись для уменьшения значения переменной на 1 выглядит так:

```
i--;
```

При выводе результата ситуация несколько усложняется, потому что нужно вывести значения трех переменных и два символа: «+» и «=». Для этого строится список вывода:

```
cout << a << \+\ << b << \=\ << c;
```

Обратите внимание, что имена переменных записаны без кавычек.

1.1.4 Типы данных

Любая переменная относится к какому-либо типу, то есть может хранить данные только того типа, который был указан при ее объявлении.

В языках C++ используются следующие основные типы данных:

— **int** - целые значения;

— **float** - вещественные значения;

— **bool** - логические значения;

— **char** - символ (в памяти хранится код символа);

На переменные типа **char** и **bool** в памяти выделяется 1 байт, на переменную типа **int** - 2 или 4 байта (в зависимости от версии языка), на переменную типа **float** - 4 байта.

Существуют также расширенные типы данных для работы с большими числами и повышения точности вычислений:

— **long int** - длинное целое число (4 или 8 байт);

— **double** - вещественное число двойной точности (8 байт) .

Логические переменные относятся к типу **bool** и принимают значения **true** (истина) и **false** (ложь). Несмотря на то, что теоретически для хранения логического значения достаточно одного бита памяти, такая переменная занимает в памяти один байт. Так как процессор может читать и записывать в память только целые байты, операции с логическими переменными в этом случае выполняются быстрее[9].

Арифметические выражения и операции

Арифметические выражения в любом языке программирования записываются в строчку. Они могут содержать константы (постоянные значения), имена переменных, знаки арифметических операций, круглые скобки (для изменения порядка действий) и вызовы функций. Например,

```
a =(c+5−1)\2*d;
```

При определении порядка действий используется *приоритет* (старшинство) операций. Они выполняются в следующем порядке:

— действия в скобках;

— умножение и деление, слева направо;

— сложение и вычитание, слева направо.

Таким образом, умножение и деление имеют одинаковый приоритет, более высокий, чем сложение и вычитание. Поэтому в приведенном примере значение выражения, заключенного в скобки, сначала разделится на 2, а потом результат деления умножится на **d**.

В языке C++ часто используют сокращенную запись арифметических операций:

```
a+=b;   a=a+b;
a-=b;   a=a-b;
a*=b;   a=a*b;
a\= b;  a=a\b;
```

Если в выражение входят переменные разных типов, в некоторых случаях происходит автоматическое приведение типа к более «широкому». Например, результат умножения целого числа на вещественное - это вещественное число.

В языке C++ результат деления целого числа на целое - это всегда **целое** число, остаток при делении отбрасывается. Когда нужно получить вещественный результат деления, одно из чисел (делимое или делитель) нужно преобразовать к вещественному типу: для числа поставить десятичную точку, а для переменной или выражения использовать явное приведение типа:

```
int a=3,
float x; b=4;
x=3/4;              /=0
x=3./4;             /=0.75
x=3/4.;           /=0.75
x=a/4;              /=0
x=a/4.;             /=0.75
x=a/b;              /=0
x=float(a)/4;     //=0.75
x=a/float(b);         //=0.75
```

Если нужно получить остаток от деления, применяют операцию «%», которая имеет такой же приоритет, как умножение и деление:

```
d=85;
a=d/10;     //=8
b=d%10;     //=5
```

Нужно учитывать, что для отрицательных чисел эти операции выполняются, строго говоря, не совсем корректно. Дело в том, что с точки зрения теории чисел остаток - это неотрицательное число, поэтому −7=(−4)*2+1, то есть частное от деления (−7) на 2 равно −4, а остаток −1.

Операции возведения в степень в C++ нет. Для вещественных чисел можно использовать функцию **pow(x,y)**, которая возводит значение **x** в степень **y**. В языке C++ подключается файл **cmath**[10].

1.1.5 Вещественные значения

При записи вещественных чисел в программе целую и дробную часть разделяют не запятой а точкой. Например:

```
float x;
x=123.456;
```

Вещественное значение можно записывать в целочисленную переменную, при этом дробная часть значения отсекается.

1.1.6 Стандартные функции

В стандартную математическую библиотеку, которая подключается с помощью команды

```
#include <cmath>
```

Математические функции:

— **abs(a)** - модуль целого числа a ;

— **fabs(x)** - модуль вещественного числа x ;

— **sqrt(x)** - квадратный корень числа x ;

— **sin(x)** - синус угла x , заданного в радианах;

— **cos(x)** - косинус угла x , заданного в радианах;

— $exp(x)$ - e^x , экспонента числа x ;

— **ln(x)** - натуральный логарифм числа x ;

— **pow(x,y)** - возведение числа x в степень y ;

Для перехода от вещественных значений к целым используется явное преобразование, при котором отсекается дробная часть:

```
a=(int)-1.6;    //= -1
a=(int)1.6;     //=1
```

Кроме того, можно использовать стандартные функции:

— **floor(x)** - округление числа x до ближайшего целого, не большего, чем x ; возвращает вещественное число;

— **ceil(x)** - округление числа x до ближайшего целого, не меньшего, чем x ; возвращает вещественное число.

```
float x;
floor(-1.6);      //=-2
x=ceil(-1.6);     //= -1
x=floor(1.6);     //= 1
x=ceil(1.6);      //=2
```

1.1.7 Случайные числа

Случайные числами называют последовательность чисел, в которой невозможно предсказать следующее число, даже зная все предыдущие.

В языке C++ существует функция **rand** для получения случайных (точнее, псевдослучайных) целых чисел в диапазоне [**0,RAND_MAX**], где **RAND_MAX** - постоянная, определенная в заголовочном файле **cstdlib**. Целое число в заданном диапазоне [a,b] можно получить с помощью операции взятия остатка и простой арифметики:

```
n= a+rand()\%(b-a+1);
```

Например, для того, чтобы записать в целую переменную **n** случайное число в диапазоне от 1 до 6 (результат бросания кубика), можно использовать оператор

```
n = 1+rand()\%6;
```

1.2 Условный оператор

В большинстве реальных задач порядок действий может несколько изменяться, в зависимости от того, какие данные поступили. Например, программа для системы пожарной сигнализации должна выдавать сигнал тревоги, если данные с датчиков показывают повышение температуры или задымленност

Для этой цели в языках программирования предусмотрены условные операторы. Например, для того, чтобы записать в переменную **M** максимальное из значений переменных **a** и **b**, можно использовать оператор:

```
if (a>b)
M=a;
else
M=b;
```

Здесь использованы два ключевых слова: **if** означает «если», а **else** - «иначе». Если условие в скобках после **if** истинно, выполняется оператор, записанный после скобок, а если условие ложно, то выполняется оператор после слова **else**.

В приведенном примере условный оператор записан в полной форме: в обоих случаях (истинно условие или ложно) нужно выполнить некоторые действия. Программа выбора максимального значения может быть написана иначе:

```
M=a;
if (b>a)
M=b;
```

Здесь использован условный оператор в неполной форме, потому что в случае, когда условие ложно, ничего делать не требуется (нет слова **else** и операторов после него).

Часто при каком-то условии нужно выполнить сразу несколько действий. Например, в задаче сортировки значений переменных **a** и **b** по возрастанию нужно поменять местами значения этих переменных, если **a>**b:

```
if (a>b)
{
c=a;
a=b;
```

```
b=c;
}
```

В этом случае нужно записать *составной оператор*, в котором между фигурными скобками может быть сколько угодно команд.

Кроме знаков <и>, в условиях можно использовать другие знаки отношений: <=(меньше или равно), >= (больше или равно), == (равно) и != (не равно)[10].

Внутри условного оператора могут находиться любые операторы, в том числе и другие условные операторы. Например, пусть возраст Сергея записан в переменной a, а возраст Влада - в переменной **b**. Нужно определить, кто из них старше. Одним условным оператором тут не обойтись, потому что есть три возможных результата: старше Сергей, старше Влад и оба одного возраста. Решение задачи можно записать так:

```
if (a>b)
cout<<"Сергей старше";
else
if (a==b)
cout<<"Одного возраста";
else
cout<<"Влад старше";
```

Условный оператор, проверяющий равенство, находится внутри блока **else**, поэтому он называется *вложенным* условным оператором. Как видно из этого примера, использование вложенных условных операторов позволяет выбрать один из *нескольких* (а не только из двух) вариантов.

Нужно помнить правило: любой блок **else** относится к ближайшему предыдущему оператору **if**, у которого такого блока еще не было.

1.3 Сложные условия

Предположим, что ООО «Мечта» набирает сотрудников, возраст которых от 25 до 40 лет включительно. Нужно написать программу, которая запрашивает возраст претендента и выдает ответ: «подходит» он или «не подходит» по этому признаку.

На качестве условия в условном операторе можно указать любое логическое выражение, в том числе сложное условие, составленное из простых отношений с помощью логических операций (связок) «И», «ИЛИ» и «НЕ».

В языке C++ они записываются так: «И» - &&, «ИЛИ» - || и «НЕ» - ! (восклицательный знак).

Пусть в переменной *v* записан возраст сотрудника. Тогда нужный фрагмент программы будет выглядеть следующим образом:

```
if (v>=25 && v<=$40 )
cout<<"подходит";
else
cout<<"не подходит";
```

Обратите внимание, что каждое простое условие не обязательно заключать в скобки. Это связано с тем, что в языке C++ отношения имеют более *высокий* приоритет, чем логические операции, которые выполняются в таком порядке: сначала все операции «НЕ», затем - «И», и в самом конце - «ИЛИ» (во всех случаях - слева направо). Для изменения порядка действий используют круглые скобки.

Множественный выбор

Условный оператор предназначен, прежде всего, для выбора одного из двух вариантов (простого ветвления). Иногда нужно сделать выбор из нескольких возможных вариантов [10].

Пусть, например, в переменной m хранится номер месяца, и нужно вывести на экран его русское название. Конечно, в этом случае можно использовать 12 условных операторов:

```
if (m==1) cout<<"январь";
if (m==2) cout<<"февраль";
...
if (m==12)cout<<"декабрь";
```

Вместо многоточия могут быть записаны аналогичные операторы для остальных значений **m**. Но в языке C++ для подобных случаев есть специальный оператор выбора:

```
switch (m) {
case 1:    cout<<"январь";
break;
case 2:    cout<<"февраль";
break;
```

```
case 12: cout<<"декабрь";
break;
default: cout<<"ошибка";
}
```

Кроме очевидных 12 блоков здесь добавлен еще один, который сигнализирует об ошибочном номере месяца. Он начинается ключевым словом **default**.

Обратите внимание, что каждый блок заканчивается оператором **break** (в переводе с английского - «прервать»). Если его не поставить, программа перейдет на следующий блок. Поэтому программа

```
switch (m) {
case 1:    cout<<"январь";
case 2:    cout<<"февраль";
case 3:    cout<<"март";
default: cout<<"ошибка";
}
```

для случая m=2 выведет на экран текст

февральмартошибка

1.4 Циклы

1.4.1 Цикл с условием

Цикл - это многократное выполнение одинаковых действий. Доказано, что любой алгоритм может быть записан с помощью трех алгоритмических конструкций: циклов, условных операторов и последовательного выполнения команд (линейных алгоритмов).

Вся суть цикла - это повторять до тех пор, пока некоторое условие не становится ложно.

Рассмотрим следующую задачу: определить количество цифр в десятичной записи целого положительного числа. Будем предполагать, что исходное число записано в переменную **n** целого типа.

Сначала составим алгоритм решения этой задачи. Чтобы считать что-то в программе, нужно использовать переменную, которую называют *счетчиком*. Для подсчета количества цифр необходимо как-то отсекать эти цифры

по одной, с начала или с конца, каждый раз увеличивая счетчик. Начальное значение счетчика должно быть равно нулю, так как до выполнения алгоритма еще не найдено ни одно цифры.

Отсечь последнюю цифру проще - достаточно разделить число нацело на 10 (поскольку речь идет о десятичной системе). Операции отсечения и увеличения счетчика нужно выполнять столько раз, сколько цифр в числе. Как же «поймать» момент, когда цифры кончатся? Несложно понять, что в этом случае результат очередного деления на 10 будет равен нулю, это и говорит о том, что отброшена последняя оставшаяся цифра. Изменение переменной n и счетчика для начального значения 1234 можно записать в виде таблицы. Запись такого цикла на языке C++ выглядит так:

```
count=0;
while (n>0)
{
n=n\10;
count ++;
}
```

1.4.2 Цикл с переменной

Здесь целочисленная переменная-счетчик имеет имя **count**. Слово **while** переводится с английского как «пока», за ним в скобках записывается условие работы цикла (в данном случае - «пока **n**>0»). Фигурные скобки ограничивают составной оператор. Если в теле цикла нужно выполнить только один оператор, эти скобки можно не ставить.

Если проверка условия выполняется в начале очередного шага цикла. Такой цикл называется циклом с предусловием (то есть с предварительной проверкой условия) или циклом «пока». Если в начальный момент значение переменной **n** будет нулевой или отрицательное, цикл не выполнится ни одного раза[10].

В данном случае количество шагов цикла «пока» неизвестно, оно равно количеству цифр введенного числа, то есть зависит от исходных данных. Кроме того, этот же цикл может быть использован и в том случае, когда число шагов известно заранее или может быть вычислено:

```
k=0;
while ( k<10 )
```

```
{
cout<<"привет\n";
k++;
}
```

Если условие в заголовке цикла никогда не нарушится, цикл будет работать бесконечно долго. В этом случае говорят, что «программа зациклилась». Например, если забыть увеличить переменную **k** в предыдущем цикле, программа зациклится:

```
k=0;
while(k<10 )
{
cout<<"привет\n";
}
```

Во многих языках программирования существует *цикл с постусловием*, в котором условие проверяется после завершения очередного шага цикла. Это полезно в том случае, когда нужно обязательно выполнить цикл хотя бы один раз. Например, пользователь должен ввести с клавиатуры положительное число. Для того, чтобы защитить программу от неверных входных данных, можно использовать такой цикл с постусловием:

```
do
{
cout<<"Введите␣n>0:␣";
cin>>n;
}
while (n<=0);
```

Этот цикл закончится тогда, когда условие **n<=0** нарушится, то есть станет *истинным* условие **n>0**. А это значит, что пользователь ввел допустимое (положительное) значение.

Обратите внимание на важную особенность этого вида цикла: при входе в цикл условие не проверяется, поэтому цикл *всегда выполняется хотя бы один раз*.

Вложенные циклы

В более сложных задачах часто бывает так, что на каждом шаге цикла нужно выполнять обработку данных, которая также представляет собой циклический алгоритм. В этом случае получается конструкция «цикл в цикле» или «вложенный цикл».

Предположим, что нужно найти все простые числа в интервале от 2 до 1000. Простейший (но не самый быстрый) алгоритм решения такой задачи на псевдокоде выглядит так:

```
сделать для n от 1 до 1000
если число n простое то
вывод n
```

Чтобы проверить делимость числа n на некоторое число **k**, нужно взять остаток от деления **n** на **k**. Если этот остаток равен нулю, то **n** делится на **k**. Таким образом, программу можно записать так (здесь **n**, **k** и **count** - целочисленные переменные, **count** обозначает счетчик делителей):

```
for(n=2; n<=1000; n++)
{
count=0;
for (k=2; k<n; k++)
if (n%k==0) count++;
if (count==0) cout<<n<<endl;
}
```

1.5 Процедуры

Предположим, что в нескольких местах программы требуется выводить на экран сообщение об ошибке: «Ошибка программы». Это можно сделать, например, так:

```
cout<<"Ошибка␣программы";
```

Конечно, можно вставить этот оператор вывода везде, где нужно вывести сообщение об ошибке. Но тут есть две сложности. Во-первых, строка-сообщение хранится в памяти много раз. Во-вторых, если мы задумаем поменять текст сообщения, нужно будет искать эти операторы вывода по всей программе. Для таких случаев в языках программирования предусмотрены процедуры - вспомогательные алгоритмы, которые выполняют некоторые // действия[10].

```
 main()
{
int n;
cin>>n;
if (n<0) Error();
...
}
```

Фактически мы ввели в язык программирования новую команду **Error**, которая была расшифрована прямо в теле программы. Для того, чтобы процедура заработала, в основной программе (или в другой процедуре) необходимо ее *вызвать* по имени.

Процедура начинается с ключевого слова **void** («пустой», то есть алгоритм, не возвращающий никакого значения). Тело процедуры заключается в фигурные скобки. Процедура расположена выше основной программы, так чтобы в момент *вызова* процедуры транслятор уже знал о ней. Обратите внимание, что и в заголовке процедуры, и при ее вывозе после имени процедуры нужно ставить круглые скобки (в данном случае - пустые).

Как мы видели, использование процедур сокращает код, если какие-то операции выполняются несколько раз в разных местах программы. Кроме того, большую программу разбивают на несколько процедур для удобства, оформляя в виде процедур отдельные этапы сложного алгоритма. Такой подход делает всю программу более понятной.

Процедура с параметрами

Процедура **Error** при каждом вызове делает одно и то же. Более интересны процедуры, которым можно передавать аргументы - данные, которые изменяют выполняемые действия. Внутри процедуры эти данные рассматриваются как внутренние (локальные) переменные и называются *параметрами*.

Предположим, что в программе требуется многократно выводить на экран запись целого числа (0..255) в 8-битном двоичном коде. Старшая цифра в такой записи - это частное от деления числа на 128. Далее возьмем остаток от этого деления и разделим на 64 - получается вторая цифра и т.д. Алгоритм, решающий эту задачу для переменной **n**, можно записать так:

```
k=128;
while ( k > 0 )
{
```

```
cout<<n/k;
n=n%k;
k=k/2;
}
```

Писать такой цикл каждый раз, когда нужно вывести двоичное число, очень утомительно. Кроме того, легко сделать ошибку или опечатку, которую будет сложно найти. Поэтому лучше оформить этот вспомогательный алгоритм в виде процедуры. Но этой процедуре нужно передать *аргумент* - число для перевода в двоичную систему. Программа получается такая:

```
void printBin (int n)
{
int k;
k=128;
while ( k>0 )
{
cout<<n/k;
n=n%k;
k=k/2;
}
main()
{
printBin (99);
}
```

Основная программа содержит всего одну команду - вызов процедуры **printBin** с аргументом 99. В заголовке процедуры в скобках записывают тип и внутреннее имя параметра (то есть имя, по которому к нему можно обращаться в процедуре).

В процедуре объявлена *локальная* (внутренняя) переменная **k** - она известна только внутри этой процедуры, обратиться к ней из основной программы и из других процедур невозможно [10].

1.6 Функции

Функция, как и процедура - это вспомогательный алгоритм, который может принимать аргументы. Но, в отличие от процедуры, функция всегда

возвращает *значение-результат*. Результатом может быть число, символ, или объект другого типа.

Составим функцию, которая вычисляет сумму цифр числа. Будем использовать следующий алгоритм (предполагается, что число записано в переменной **n**):

```
сумма=0
пока n!=0
{
сумма=сумма+n%10
n=n/10;
}
```

Чтобы получить последнюю цифру числа (которая добавляется к сумме) нужно взять остаток от деления числа на 10. Затем последняя цифра отсекается, и мы переходим к следующей цифре. Цикл продолжается до тех пор, пока значение **n** не становится равно нулю.

В языке C++ используют специальный оператор **return** (от англ. «вернуть»), после которого записывают значение-результат:

```
int sumDigits (int n)
{
int sum=0;
while ( n!=0 )
{
sum+=n%10;
n\=10;
}
return sum;
}
```

Тип возвращаемого значения **(int)** указывается в заголовке функции перед именем функции. Так же как и в процедурах, в функциях можно объявлять и использовать локальные переменные. Они входят в «зону видимости» только этой функции, для всех остальных функций и процедур они недоступны.

В функции может быть несколько операторов **return**, после выполнения любого из них работа функции заканчивается.

Функции, созданные в программе таким образом, применяются точно так же, как и стандартные функции. Их можно вызывать везде, где может использоваться выражение того типа, который возвращает функция.

1.7 Логические функции

Достаточно часто применяют специальный тип функций, которые возвращают *логическое значение* (**да** или **нет**, **true** или **false**). Иначе говоря, такая *логическая* функция отвечает на вопрос «да или нет?» и возвращает 1 бит информации.

Вернемся к задаче, которую мы уже рассматривали: вывести на экран все простые числа в диапазоне от 2 до 1000. Алгоритм определения простоты числа оформим в виде функции. При этом его можно будет легко вызвать из любой точки программы.

Запишем основную программу на псевдокоде:

```
для i от 2 до 100
если i – простое то
вывод i
```

Предположим, что у нас уже есть логическая функция **isPrime**, которая определяет простоту числа, переданного ей как параметр, и возвращает **true** («истинно»), если число простое, и **false** («ложно») в противном случае. Такую функцию можно использовать вместо выделенного блока алгоритма:

```
bool isPrime (int n)
{
int k=2;
while (k*k<=n&&n%k!=0 ) k++;
return (k*k>n );
}
```

Эта функция возвращает логическое значение, на это указывает ключевое слово bool (от boolean - булевский, логический, в честь Дж. Буля).

Для того, чтобы использовать переменные типа bool в языке C++, в начале программы нужно подключить файл stdbool.h с помощью директивы **# include**.

В этом файле константе **true** присваивается значение 1, а константе **false** - значение 0. Дело в том, что в языке C++ любое значение, отличное от

нуля, соответствует истинному условию, а нулевое значение - ложному. Поэтому логическая функция может возвращать и целое значение - 1 как ответ «да» и 0 как ответ «нет». Форма вызова такой функции не меняется [10].

Логические функции можно использовать так же, как и любые условия: в условных операторах и циклах с условием. Например, такой цикл останавливается на первом введенном составном числе:

```
cin>>n;
while (isPrime(n))
{
cout<<n<<" простое число"<<endl;
cin>>n;
}
```

1.8 Рекурсия.

Вспомним определение натуральных чисел. Оно состоит из двух частей:

1 - натуральное число;

2 - если n - натуральное число, то n+1 - тоже натуральное число.

Вторая часть этого определения в математике называется *индуктивным*: натуральное число определяется через другое натуральное число, и таким образом определяется все множество натуральных чисел. В программировании этот прием называют *рекурсией*.

Рекурсия - это способ определения множества объектов через само это множество на основе заданных простых базовых случаев.

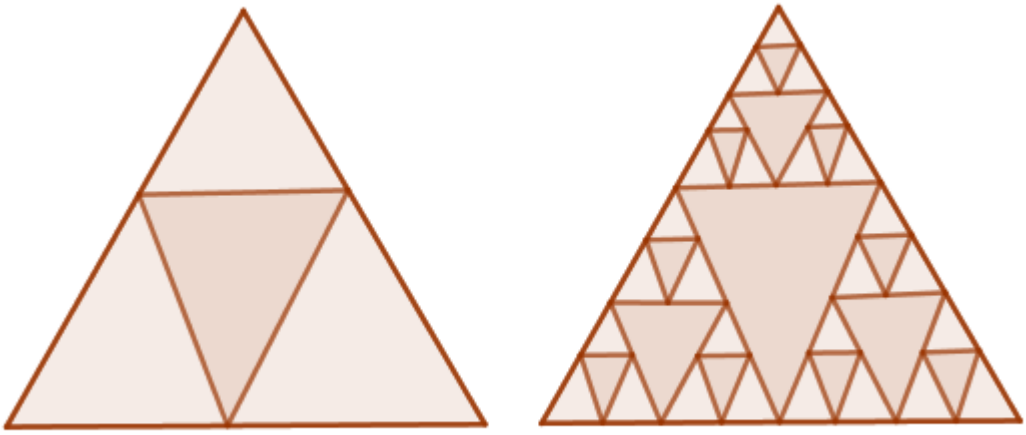
Первая часть в определении натуральных чисел - это и есть тот самый базовый случай. Если убрать первую часть из определения, оно будет неполно: вторая часть дает только метод перехода к следующему значению, но не дает никакой «зацепки», не отвечает на вопрос «откуда начать».

Похожим образом задаются числа Фибоначчи: первые два числа равны 1, а каждое из следующих чисел равно сумме двух предыдущих:

1) $F_1 = F_2 = 1$,

2) $F_n = F_{n-1} + F_{n-2}$ для n>2 .

Популярные примеры рекурсивных объектов - *фракталы*. Так в математике называют геометрические фигуры, обладающие *самоподобием*. Это значит, что они составлены из фигур меньшего размера, каждая из которых подобна целой фигуре.



На рисунке показан треугольник Серпинского - один из первых фракталов, который предложил в 1915 году польский математик В. Серпинский.

Равносторонний треугольник делится на 4 равных треугольника меньшего размера (левый рисунок), затем каждый из полученных треугольников, кроме центрального, снова делится на 4 еще более мелких треугольника и т.д. На правом рисунке показан треугольник Серпинского с тремя уровнями деления.

Пример. Составим функцию, которая вычисляет сумму цифр числа. Будем рассуждать так: сумма цифр числа **n** равна значению последней цифры плюс сумма цифр числа **n/10**. Сумма цифр однозначного числа равна самому этому числу, это условие окончания рекурсии. Получаем следующую функцию:

```
int sumDig (int n)
{
int sum;
sum=n%10;
if (n>=10)
sum+=sumDig (n/10);
return sum;
}
```

1.9 Массивы. Перебор элементов массива

Основное предназначение современных компьютеров - обработка большого количества данных. При этом надо как-то обращаться к каждой из ты-

сяч (или даже миллионов) ячеек с данными. Очень сложно дать каждой ячейке собственное имя и при этом не запутаться. Из этой ситуации выходят так: дают имя не ячейке, а группе ячеек, в которой каждая ячейка имеет собственный номер. Такая область памяти называется массивом.

Массив - это группа переменных одного типа, расположенных в памяти рядом (в соседних ячейках) и имеющий общее имя. Каждая ячейка в массиве имеет уникальный номер[10].

Для работы с массивами нужно, в первую очередь, научиться:

— выделять память нужного размера под массив;

— записывать данные в нужную ячейку;

— читать данные из ячейки массива.

Чтобы использовать массив, надо его объявить - определить тип массива (тип входящих в него элементов), выделить место в памяти и присвоить имя. Имена массивов строятся по тем же правилам, что и имена переменных.

В языке C++ массивы объявляются почти так же, как и обычные переменные, только после имени массива в квадратных скобках указывают количество элементов:

```
int A[5];
double V[8];
bool L[10];
char S[80];
```

Индексы элементов массива всегда начинаются с нуля. Если, например, в массиве A пять элементов, то последний элемент будет иметь индекс 4.

Для того, чтобы обратиться к элементу массива, нужно записать имя массива и в квадратных скобках индекс нужного элемента, например, **A[3]**. Индексом может быть не только число, но значение целой переменной или арифметического выражения целого типа. В этом примере массив заполняется квадратами первых натуральных чисел:

```
main()
{
const int N=10;
int A[N];
int i;
for (i=0; i<N; i++ )
A[i]=i*i;
}
```

При объявлении границ индексов массивов можно использовать *константы* - постоянные величины, имеющие имя. В приведенном примере с помощью ключевого слова **const** объявлена целочисленная **(int)** константа **N**, равная 10. Константы обычно вводятся выше блока объявления переменных. Использование констант очень удобно, потому что при изменении размера массива в программе нужно поменять только одно число - значение этой константы[10].

Далее во всех примерах мы будем считать, что в программе объявлен целочисленный массив **A**, состоящий из **N** элементов (с индексами от 0 до **N-1**), а также целочисленная переменная **i**, которая будет обозначать индекс элемента массива. Чтобы ввести такой массив или вывести его на экран, нужно использовать цикл, то есть ввод и вывод массива выполняется поэлементно:

```
for (i=0; i<N; i++ )
{
cout<<"A["<<i<<"]=";
cin>>A[i];
}
for (i=0; i<N; i++)
cout<<A[i]<<"␣";
```

В этом примере перед вводом очередного элемента массива на экран выводится подсказка. Например, при вводе 3-го элемента будет выведено «**A[3]=**». После вывода каждого элемента ставится пробел, иначе все значения сольются в одну строку.

В учебных примерах массивы часто заполняют случайными числами:

```
for (i=0; i<N; i++)
{
A[i]=irand (20, 100);
cout<<A[i]<<"␣";
}
```

1.10 Отбор элементов массива по условию.

Перебор элементов состоит в том, что мы в цикле просматриваем все элементы массива и, если нужно, выполняем с каждым из них некоторую опе-

рацию. Для этого удобнее всего использовать цикл с переменной, которая изменяется от минимального до максимального индекса. Поскольку мы будем работать с массивом, элементы которого имеют индексы от 0 до **N-1**, цикл выглядит так:

```
for (i=0; i<N; i++)
{
...
}
```

Здесь вместо многоточия можно добавлять операторы, работающие с элементом **A[i]**.

Во многих задачах нужно найти в массиве все элементы, удовлетворяющие заданному условию, и как-то их обработать. Простейшая из таких задач - подсчет нужных элементов. Для решения этой задачи нужно ввести переменную-счетчик, начальное значение которой равно нулю. Далее в цикле (от 0 до **N-1**) просматриваем все элементы массива. Если для очередного элемента выполняется заданное условие, то увеличиваем счетчик на 1. На псевдокоде этот алгоритм выглядит так:

```
счетчик=0;
for (i=0; i<N; i++ )
if (условие выполняется для A[i])
счетчик ++;
```

Предположим, что в массиве **A** записаны данные о росте игроков баскетбольной команды.

Найдем количество игроков, рост которых больше 180 см, но меньше 190 см. В следующей программе используется переменная-счетчик **count**:

```
count=0;
for (i=0; i<N; i++)
if ( 180<A[i] && A[i]<190 )
count ++;
```

1.11 Сортировка массивов.

Сортировка-это расстановка элементов массива в заданном порядке.

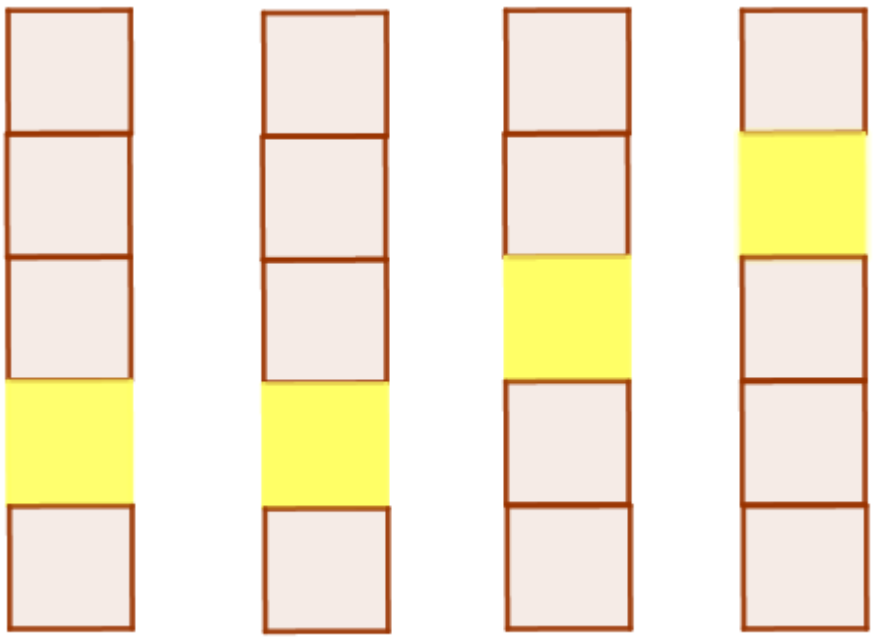
Порядок сортировки может быть любым; для чисел обычно рассматривают сортировку по возрастанию (или убыванию) значений.

В целом их можно разделить на две группы: 1) простые, но медленно работающие (на больших массивах) и 2) сложные, но быстрые.

Метод пузырька (сортировка обменами)

Название этого метода произошло от известного физического явления - пузырек воздуха в воде поднимается вверх. Если говорить о сортировке массива, сначала поднимается «наверх» (к началу массива) самый «легкий» (минимальный) элемент, затем следующий и т.д.[10]

Сначала сравниваем последний элемент с предпоследним. Если они стоят неправильно (меньший элемент «ниже»), то меняем их местами. Далее так же рассматриваем следующую пару элементов и т.д. (см. рисунок).



Когда мы обработали пару (**A[0]**, **A[1]**), минимальный элемент стоит на месте **A[0]**. Это значит, что на следующих этапах его можно не рассматривать. Первый цикл, устанавливающий на свое место первый (минимальный) элемент, можно на псевдокоде записать так:

```
для j от N–2 до 0 шаг –1
если A[j+1]<A[j] то
поменять местами A[j] и A[j+1]
```

Здесь **j** - целочисленная переменная. Обратите внимание, что на очередном шаге сравниваются элементы **A[j]** и **A[j+1]**, поэтому цикл начинается с **j=N-2**. Если начать с **j=N-1**, то на первом же шаге получаем выход за границы массива - обращение к элементу **A[N]**.

За один проход такой цикл ставит на место один элемент. Чтобы «подтянуть» второй элемент, нужно написать еще один почти такой же цикл, который будет отличаться только конечным значением **j** в заголовке цикла. Так как верхний элемент уже стоит на месте, его не нужно трогать:

```
сделать для j от N–2 до 1 шаг –1
если A[j+1]<A[j] то
поменять местами A[j] и A[j+1]
```

При установке 3-го элемента конечное значение для **j** будет равно 2 и т.д.

Таких циклов нужно сделать **N-1** - на 1 меньше, чем количество элементов массива. Почему не **N**? Дело в том, что если **N-1** элементов поставлены на свои места, то оставшийся автоматически встает на свое место - другого места для него нет. Поэтому полный алгоритм сортировки представляет собой такой вложенный цикл:

```
for ( i=0; i<N–1; i++)
for ( j=N–2; j>=i; j––)
if ( A[j]>A[j+1])
{
// поменять местами A[j] и A[j+1]
}
```

Метод выбора

Еще один популярный простой метод сортировки - метод выбора, при котором на каждом этапе выбирается минимальный элемент (из оставшихся) и ставится на свое место. Алгоритм в общем виде можно записать так:

```
для i от 1 до N–1
найти номер nMin минимального элемента из A[i]..A[N]
если i!=nMin то
поменять местами A[i] и A[nMin]
```

Здесь перестановка происходит только тогда, когда найденный минимальный элемент стоит не на своем месте, то есть **i**! =**nMin**. Поскольку поиск минимального элемента выполняется в цикле, этот алгоритм сортировки также представляет собой вложенный цикл:

```
for (i=0; i<N-1; i++) { nMin=i;
for (j=i+1; j<N; j++) if (A[j]<A[nMin])
nMin=j;
{
// поменять местами A[i] и A[nMin]
}
}
```

1.12 Символьные строки

Символьная строка - это последовательность символов, расположенных в памяти рядом (в соседних ячейках). Для работы с символами во многих языках программирования есть переменные специального типа: символы и символьные массивы. Казалось бы, массив символов - это и есть символьная строка, однако здесь есть некоторые особенности.

Дело в том, что массив - это группа символов, каждый из которых независим от других. Это значит, что вводить символьный массив нужно посимвольно, в цикле. Более того, размер массива задается при объявлении, и не очень ясно, как использовать массивы для работы со строками переменной длины. Таким образом, нужно

— работать с целой символьной строкой как с единым объектом;

— использовать строки переменной длины.

Для работы с символьными строками в C++ введен специальный тип данных, который называется **string**[10]:

```
main()
{
string s;
...
}
```

Начальное значение строки можно задать прямо при объявлении:

```
string s = "Привет!";
```

Новое значение строки записывается с помощью оператора присваивания:

```
s="Привет!";
```

Для того, чтобы ввести из входного потока строку с пробелами, применяется функция **getline**:

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

а вывод выполняется стандартным образом:

```
cout<<s;
```

Для определения длины строки s используется запись **s.size()**. Это так называемая точечная нотация. Такая запись означает, что метод **size** применяется к объекту s типа **string**. В данном случае **size** - это функция (метод), связанная с типом данных **string**.Таким образом, программа которая заменяет в строке все буквы ’а’ на буквы ’б’ на языке C++ выглядит так:

```
#include <iostream>
using namespace std;
main()
{
string s;
int i;
cout<<"Введите␣строку:␣";
getline (cin, s);
for (i=0; i<s.size(); i++)
if (s[i]=='a')
s[i]='б'; cout<<s;
}
```

1.13 Функции для работы с символьными строками

1.13.1 Операции со строками

В языке C++ операции со строками выполняются значительно проще благодаря введенному типу **string**.

Оператор '+' используется для объединения (сцепления) строк, эта операция иногда называется *конкатенация*. Например:

```
s1="Привет";
s2="Мир";
s=s1+",␣"+s2+"!";
```

Здесь и далее считаем, что в программе объявлены строки **s, s1 и s2**. В результате выполнения приведенной программы в строку s будет записано значение **"Привет, Мир!"**[10].

Для того, чтобы выделить часть строки (*подстроку*, англ. *substring*), применяется метод **substr**, который тоже вызывается с помощью точечной нотации. Этот метод принимает два параметра: номер начального символа и *количество символов*. Следующий фрагмент копирует в строку **s1** пять символов строки **s** (с 3-го по 7-й):

```
s="0123456789";
s1=s.substr (3,5);
cout<<s1<<endl;
```

В строку **s1** будет записано значение «**34567**». Если второй параметр при вызове **substr** не указан, метод возвращает все символы до конца строки. Например:

```
s="0123456789";
s1=s.substr(3);
```

вернет «**3456789**».

Для удаления части строки нужно вызвать метод **erase**, указав номер начального символа и число удаляемых символов:

```
s="0123456789";
s.erase (3, 6);
```

В строке **s** остается значение «**0129**» (удаляются 6 символов, начиная с 3-го). Обратите внимание, то процедура **erase** изменяет строку.

При вставке символов методу **insert** передают вставляемый фрагмент и номер символа, с которого начинается вставка:

```
s="0123456789";
s.insert (3, "ABC");
```

Переменная **s** получит значение «**012ABC3456789**».

1.13.2 Поиск в строках

В языке C++ для поиска символа и подстроки используется метод **find**. Эта функция возвращает номер найденного символа (номер первого символа подстроки) или -1, если найти нужный фрагмент не удалось.

```
string s="Здесь␣был␣Вася.";
int n;
n=s.find ('c'); if (n>=0)
cout<<"Номер␣первого␣символа␣'c':"<<n<<endl;
else cout<<"Символ␣не␣найден.\n";
n=s.find ("Вася");
if (n>=0 )
cout<<"Слово␣'Вася'␣начинается␣с␣s["<<n<<"]\n";
else
cout<<"Слово␣не␣найдено.\n";
```

Для поиска с конца строки используют метод **rfind**:

```
n=s.rfind ('c');
if (n>=0)
cout<<"Номер␣последнего␣символа␣'c':␣"<<n<<endl;
else
cout<<"Символ␣не␣найден.\n";
```

Пример обработки строк

Предположим, что с клавиатуры вводится строка, содержащая имя, отчество и фамилию человека, например:

Васила Александрова Маликова

Каждые два слова разделены одним пробелом, в начале строки пробелов нет. В результате обработки должна получиться новая строка, содержащая фамилию и инициалы:

Маликова В.А.

```
ввести строку s
найти в строке s первый пробел
имя=все, что слева от первого пробела
удалить из строки s имя с пробелом
найти в строке s первый пробел
отчество=все, что слева от первого пробела
удалить из строки s отчество с пробелом
// осталась фамилия
s=s+' '+имя[1]+'.'+отчество[1]+'.'
```

Мы последовательно выделяем из строки три элемента: имя, отчество и фамилию, используя тот факт, что они разделены одиночными пробелами. После того, как имя сохранено в отдельной переменной, в строке s остались только отчество и фамилия. После «изъятия» отчества остается только фамилия. Теперь нужно собрать строку-результат из частей: «сцепить» фамилию и первые буквы имени и отчества, поставив пробелы и точки между ними. Для выполнения всех операций будем использовать стандартные функции, описанные выше.

```
cout<<"Введите имя, отчество и фамилию: ";
getline (cin, s);
n=s.find(' ');
name=s.substr(0,1)+'.';
s=s.substr (n+1);
n=s.find(' ');
name2=s.substr(0,1)+'.';
s=s.substr (n+1);
s=s+' '+name+name2;
cout<<s;
```

1.13.3 Строки в процедурах и функциях

Строки можно передавать в процедуры и функции как аргументы. Построим процедуру, которая заменяет в строке s все вхождения слова-образца wOld на слово-замену **wNew** (здесь **wOld** и **wNew** - это имена переменных, а выражение «слово **wOld**» означает «слово, записанное в переменную **wOld**»).

Процедура на языке C++ выглядит просто благодаря более развитым средствам работы со строками:

```
void replaceAll (string &s, string wOld, string wNew)
                {
                string res="";
                int p, len=wOld.size();
                while (s.size()>0)
                {
                p=s.find (wOld); if (p<0)
                {
                res=res+s;
                break;
                }
                if (p>0 )
                res=res+s.substr (0,p);
                res=res+wNew;
                if (p+len>s.size()) s="";
                else s.erase (0,p+len);
                }
                s=res;
                }
```

Поскольку строка s изменяется, она передается по ссылке и в заголовке процедуры перед ее именем ставится знак &. Переменная **len** хранит длину строки-образца, в остальном алгоритм не изменяется.

Построенную выше процедуру на языке C++ можно легко превратить в функцию. Для этого нужно

— в заголовке функции указать, что она возвращает строку (использовать ключевое слово **string** вместо **void**);

— убрать в заголовке процедуры символ & перед именем исходной строки (она не должна изменяться);

— вернуть результат с помощью оператора **return**.

Ниже показаны все измененные части подпрограммы:

```
string replaceAll ( string s, string wOld, string wNew )
                {
                ...
                return res;
                }
```

Вызывать функцию можно таким образом:

```
                main()
                {
                string s="12.12.12";
                s=replaceAll (s, "12", "A12B");
                cout<<s;
                cin.get();
                }
```

1.13.4 Рекурсивный перебор

В алфавите языка племени «тумба-юмба» четыре буквы: «Ы», «Ш», «Ч» и «О». Нужно вывести на экран все слова, состоящие из L букв, которые можно построить из букв этого алфавита.

Это типичная задача на перебор вариантов, которую удобно свести к задаче меньшего размера. Будем определять буквы слова последовательно, одну за другой. Первая буква может быть любой из четырех букв алфавита. Предположим, что сначала первой мы поставили букву '**Ы**'. Тогда для того, чтобы получить все варианты с первой буквой '**Ы**', нужно перебрать все возможные комбинации букв на оставшихся **L-1** позициях.

Далее поочередно ставим на первое место все остальные буквы, повторяя процедуру.

```
                перебор L символов
w[0]='Ы'; перебор последних L-1 символов
w[0]='Ш'; перебор последних L-1 символов
w[0]='Ч'; перебор последних L-1 символов
w[0]='О'; перебор последних L-1 символов
```

Здесь через **w** обозначена символьная строка, в которой хранится рабочее слово. Таким образом, задача для слов длины **L** свелась к 4-м задачам для слов длины **L-1**. Как вы знаете, такой прием называется *рекурсией*, а процедура - рекурсивной.

Когда рекурсия должна закончиться? Тогда, когда все символы расставлены, то есть количество установленных символов **N** равно **L**. При этом нужно вывести получившееся слово на экран и выйти из процедуры.

Подсчитаем количество всех возможных слов длины **L**. Очевидно, что слов длины 1 всего 4. Добавляя еще одну букву, получаем 4*4=16 комбинаций, для трех букв - 4*4*4=64 слова и т.д. Таким образом, слов из **L** букв может быть 4^L.

Напишем программу.

```
void TumbaWords(string A, string &w, int N)
        {
        int i;
        if ( N==w.size())
        {
        cout<<w<<endl;
        return;
        }
        for (i=1; i<A.size(); i++)
        {
        w[N]=A[i];
        TumbaWords (A, w, N+1);
        }
        }
        main()
        {
        string word = "...";
        TumbaWords ( "ЫШЧЮ", word, 0 );
        cin.get();
        }
```

Обратите внимание, что параметр w (строка-результат) - это изменяемый параметр.

1.14 Сравнение и сортировка строк

Строки, как и числа, можно сравнивать. Для строк, состоящих из одних букв (русских или латинских), результат сравнения очевиден: меньше будет та строка, которая идет раньше в алфавитном порядке. Например, слово «паровоз» будет «меньше», чем слово «пароход»: они отличаются в пятой букве «в»<«х». Более короткое слово, которое совпадает с началом более длинного, тоже будет стоять раньше в алфавитном списке, поэтому «пар»<«парк»[10].

При сравнении строк используются коды символов. Тогда получается, что

«ПАР»<«Пар»<«пар».

Возьмем пару «ПАР» и «Пар». Первый символ в обоих словах одинаков, а второй отличается - в первом слове буква заглавная, а во втором - такая же, но строчная. В таблице символов заглавные буквы стоят раньше строчных, и поэтому имеют меньшие коды. Поэтому «А»<«а», «П»<«а» и «ПАР»<«Пар»<«пар».

Цифры же стоят в кодовой таблице по порядку, причем раньше, чем латинские буквы; латинские буквы - раньше, чем русские; заглавные буквы (русские и латинские) - раньше, чем соответствующие строчные. Поэтому

«5STEAM»<«STEAM»<«Steam»<«steam»<«ПАР»<«Пар»<«пар».

Сравнение строк используется, например, при сортировке. Рассмотрим такую задачу: ввести с клавиатуры 10 фамилий и вывести их на экран в алфавитном порядке.

В языке C++, где есть тип данных **string**, программа выглядит просто и понятно:

```
 main()
{
const int N=10; string s1, S[N]; int i, j;
cout<<"Введите строки:\n"; for (i=0; i<N; i++)
        getline (cin, S[i]);
        for (i=0; i<N-1; i++) for (j=N-2; j>=i; j--)
        if (S[j]>S[j+1])
        {
        s1=S[j];
        S[j]=S[j+1];
        S[j+1]=s1;
        }
```

```
cout<<"После сортировки:\n";
for (i=0; i<N; i++)
cout<<S[i]<<endl;
}
```

1.15 Матрицы

Многие программы работают с данными, организованными в виде таблиц. Например, при составлении программы для игры в крестики-нолики нужно запоминать состояние каждой клетки квадратной доски. Можно поступить так: пустым клеткам присвоить код -1, клетке, где стоит нолик - код 0, а клетке с крестиком - код 1. Тогда информация о состоянии поля может быть записана в виде таблицы:

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

Такие таблицы называются *матрицами* или двухмерными массивами. Каждый элемент матрицы, в отличие от обычного (линейного) массива имеет два индекса - номер строки и номер столбца.

Матрица-это прямоугольная таблица, составленная из элементов одного типа (чисел, строк и т.д.). Каждый элемент матрицы имеет два индекса - номера строки и столбца.

При объявлении матриц в двух парах квадратных скобок указывают два размера (количество строк и количество столбцов)

```
const int N=3, M=4;
int A[N][M];
double X[10][12];
```

Каждому элементу матрицы можно присвоить любое значение, допустимое для выбранного типа данных. Поскольку индексов два, для заполнения матрицы нужно использовать вложенный цикл. Далее в примерах будем считать, что объявлена матрица из **N** строк и **M** столбцов, а **i** и **j** - целочисленные переменные, обозначающие индексы строки и столбца. В этом примере матрица заполняется случайными числами и выводится на экран:

```
for (i=0; i<N; i++)
{
for (j=0; j<M; j++)
{
A[i][j]=irand (20, 80);
cout.width(3);
cout<<A[i][j];
}
```

Такой же двойной цикл нужно использовать для перебора всех элементов матрицы. Вот как вычисляется сумма всех элементов:

```
sum=0;
for (i=0; i<N; i++)
for (j=0; j<M; j++)
sum+=A[i][j];
```

Обработка элементов матрицы

Покажем, как можно обработать (например, сложить) некоторые элементы квадратной матрицы **A**, содержащей **N** строк и **N** столбцов.

Главная диагональ - это элементы **A[0][0], A[1][1],..., A[N-1][N-1]**, то есть номер строки равен номеру столбца. Для перебора этих элементов нужен один цикл:

```
for (i=0; i<N; i++)
{
// работаем с
}
```

Элементы побочной диагонали - это **A[0][N-1], A[1][N-2], ..., A[N-1][0]**. Заметим, что сумма номеров строки и столбца для каждого элемента равны **N-1**, поэтому получаем такой цикл перебора:

```
for (i=0; i<N; i++)
{
A[i][N–1–i]
}
```

В случае обработки всех элементов на главной диагонали и под ней нужен вложенный цикл: номер строки будет меняться от 0 до N-1, а номер столбца для каждой строки i - от 0 до i:

```
for ( j = 0; j<M; j++ )
{
c=A[2][j];
A[2][j]=A[4][j];
A[4][j]=c;
}
```

1.16 Структуры (записи)

Структурой называют тип данных, включающий в себя несколько элементов полей, т.е. элементов разных типов. (и другие структуры)

Мы работали только с простыми типами данных. Структуры, нужно тоже объявлять. Так как при объявлении переменных и массивов указывается их тип, для того, чтобы работать со структурами, нужно ввести новый тип данных.

Построим структуру, описывающий книгу в базе библиотек.

Будем хранить в структуре только

— фамилию автора (строка, менее 40 символов);

— название книги (строка, менее 80 символов);

— имеющееся в библиотеке количество экземпляров (целое число). В языке C++ для хранения символьных строк можно использовать тип **string**[10].

```
typedef struct
{
string author;// автор, строка
string title;// название, строка
int count;          // количество, целое
} TBook;
```

В результате такого объявления никаких структур в памяти не создается: мы просто описали новый тип данных, чтобы транслятор знал, что делать, если мы захотим его использовать.

Для структуры на языке C++, в которой для хранения полей **author** и **title** используется тип **string**. Команды вывода

```
cout<<sizeof(TBook)<<endl;
cout<<sizeof(B)<<endl;
cout<<sizeof(Books)<<endl;
```

покажут на экране 12, 12 и 1200.

Дело в том, что объект класса **string** фактически представляет собой указатель - ссылку на область памяти, где хранятся данные этого объекта (длина строки и все ее символы). Поэтому функция **sizeof** определяет размер именно этого указателя, который равен 4 байтам. Таким образом, использование типа **string**, с одной стороны, значительно облегчает работу с символьными строками в языке C++, а с другой может привести к неожиданным проблемам из-за того, что внутреннее устройство таких строк скрыто от программиста.

Обращение к полю структуры

Если в структуре на языке C++ используются строки типа string, в результате

```
cout<<sizeof(B.author)<<endl;
cout<<sizeof(B.title)<<endl;
cout<<sizeof(B.count)<<endl;
```

выполнения операторов: мы увидим 4, 4 и 4.

С полями структуры можно обращаться так же, как и с обычными переменными соответствующего типа. Можно вводить их с клавиатуры (или из файла):

```
cin>>B.author;
cin>>B.title;
cin>>B.count;
```

присваивать новые значения:

```
B.author="Толстой␣Л.Н.";
B.title="Юность";
B.count=1;
```

и выводить на экран.

```
cout<<B.author<<""<<B.title<<""<<B.count<<"шт.";
```

1.17 Динамические массивы

Когда мы объявляем массив, место для него выделяется во время трансляции, то есть до выполнения программы. Такой массив называется статическим. В то же время иногда размер данных заранее неизвестен.

Например, в каком то файле A есть n количество чисел. Нам их нужно отсортировать, тогда у нас есть 2 варианта: 1) выделить сразу большой блок памяти, и 2) выделить память уже *во время выполнения программы* (то есть динамически), когда точно знаем размер массива[11].

Рассмотрим другой пример: в файле находится список слов. Нужно вывести в другой файл все различные слова, которые встречаются в файле, и определить, сколько раз встречается каждое слово. Здесь проблема состоит в том, что нужный размер массива можно узнать только тогда, когда все различные слова будут найдены и таком образом задача решена. Поэтому нужно сделать так, чтобы массив мог «расширяться» в ходе работы программы.

От этих задач мы и приходим к понятию «динамических структур данных», которые в свою очередь позволяют нам:

— создать другие (новые) объекты в памяти;

— влиять на их размер;

— управлять ими (сортировать, удалять).

Память, выделяемый под них, называют «*кучей*» (англ. *heap*).

Размещение в памяти

Задача. *Введите N - целых чисел, выведите на экран в виде их возрастания.*

Как мы знаем, для хранения чисел, нам нужно завести массив, в которым будут находиться наши числа. Итак алгоритм на псевдокоде записывается так:

```
прочитать числа из массива
отсортировать наши числа по возрастанию
вывести массив чисел на экран
```

Для выделения памяти во время работы программы в языке C++ используют указатели. например, если необходим новый массив целых чисел, нужно объявить указатель на целую переменную:

```
int *A;
```

Это еще не массив, поскольку память для его элементов не выделена. Чтобы выделить память на N элементов, в языке C используется функция **new** (для ее использования нужно подключить заголовочный файл **stdlib.h** - стандартную библиотеку):

```
A=new int[N];
```

Функции **new** передается два аргумента: количество элементов массива и размер одного элемента[11].

После выделения памяти массив можно использовать так же, как обычный статический массив.

Как только динамический массив стал не нужен, можно освободить выделенную память: оператором **delete**:

```
delete[]A;
```

Квадратные скобки после оператора delete в языке C++ обозначают, что удаляется массив, а не одна переменная.

В языке C++ удобно использовать класс **vector** из стандартной библиотеки шаблонов (STL = *Standard Template Library*), который фактически представляет собой динамический массив с возможностью расширения. Такой массив (вектор) для целых чисел (элементов типа int) можно объявить так:

```
vector <int> A;
```

Количество элементов в массиве можно определить с помощью метода **size**:

```
cout<<A.size();
```

В самом начале в массиве нет ни одного элемента и его размер равен нулю. Метод *push_back* добавляет новый элемент в конец массива:

```
for (i=0; i<N; i++)
A.push\_back(i+1);
```

Работать с вектором можно так же, как и с обычным массивом:

```
for (i=0; i<A.size(); i++)
cout<<A[i]<<" ";
```

Для работы с классом vector нужно подключить заголовочный файл **vector**:

```
#include <vector>
```

1.18 Динамические матрицы

Похожим образом можно работать и с динамическими матрицами. Матрица - это «массив массивов», каждую ее строку можно рассматривать как линейный массив.

В языке C++ для работы с динамической матрицей можно использовать тип **vector** из библиотеки STL. Так как матрица - это массив из массивов, можно составить вектор, элементами которого будут вектора (массивы):

```
vector:
typedef vector<int> vint;
vector <vint> A;
```

Как видим, в первой строке вводится новый тип данных **vint**, этот тип данных называется вектором целых чисел, а вторая строка объявляет вектор с именем **A**, который состоит из элементов типа **vint**[11].

В первую очередь методом **resize** (от англ. *resize* - изменить размер) мы должны определить количество строк матрицы:

```
A.resize (N);
```

вторым действием устанавливаем размеры для каждой строки, они, как мы види, могут быть разные:

```
for (i=0; i<N; i++)
A[i].resize (M);
```

После установки размера строки можем использовать обычный массив:

```
for (i=0; i<N; i++)
for (j=0; j<M; j++)
A[i][j]=i+j;
```

Расширение массива

Задача. *Введите натуральные цифры, ввод должен заканчиваться нулем, и выведите на экран числа в порядке их возрастания*

Похожую задачу мы уже решали, и в этом случае необходимо сохранить в массиве. Но возникает проблема, что массив мы будем знать только тогда, когда уже введем все числа.

Первой идеей для решения данной задачи является добавления в массив 1 элемента и введенное число записать в последний элемент массива:

```
прочитать x
пока x!=0
{
расширить массив на 1 элемент записать x в последний элемент
прочитать x
}
```

Где **x** является целой переменной. Но при таком решении задачи все значения будут сохраняться.

К счастью в языке C++ мы имеем простой метод решений данной задачи, это использовать тип **vector** из стандартной библиотеки C++. Где массив автоматически расширяется расширяется при введении нового элемента[11].

Сейчас, когда наши числа находятся в массиве, мы можем отсортировать их любым способом, например, «пузырька» или с помощью «быстрой сортировки».

1.19 Списки

Задача. *В неком файле есть несколько слов, некоторые из которых повторяются. Каждое слово находится в отдельной строке. Постройте алфавитно-частотный словарь, в котором все разные слова должны быть записаны в другой файл по порядку, и где справа нахотся числа, которые показывают число повторения этих слов.*

Структура решения должна быть такова: «слово - количество», т.е мы должны создать список, где будут храниться эти пары.

Списком называю элементы одного типа, которые упорядоченные между собой, и для них должны быть введены операции вставки (включения) и удаления (исключения).

Для таких случаем будем использовать *линейные* списки, в котором мы можем для каждого элемента указать следующий (предыдущий).

Итак, в нашей задаче, создании алфавитно-частотного словаря, алгоритм в псевдокоде должен иметь вид:

```
            пока имеются слова в текущем файле
            {
прочитать следующее слово если оно имеется то
увеличить на 1 счетчик для этого слова
            иначе
            {
добавить слово в наш список записать 1 в счетчик слова
            }
```

1.20 Использование модулей

Как же разрабатывать большие программы? Для этого для каждого программиста необходимо дать свой блок, чтоб он работал только со своим блоком, но в свою очередь эти блоки были связаны между собой. В нашей программе в отдельный модуль можно вынести все операции со списком слов[11].

Стандартная библиотека шаблонов (STL) содержит тип данных **map** - так называемый *ассоциативный массив* или *словарь*, в котором индексом элемента может быть не только число, но и любой другой тип данных, например, символьная строка. В задаче построения алфавитно-частотного словаря элемент массива - целое число (тип **int**), а искать его нам нужно по символьной строке типа **string** (слову). В этом случае ассоциативный массив объявляется так:

```
map<string,int>L;
```

Для работы с ассоциативными массивами нужно подключить заголовочный файл **map**:

```
#include <map>
```

Что можно сделать с таким массивом? Во-первых, можно определить, сколько раз строка уже встречалась:

```
int p=L.count (s);
```

Значение переменной **p** будет равно 0, если этого слова еще нет в словаре и 1, если оно уже было добавлено. Для того, чтобы увеличить счетчик для данного слова, можно обратиться к элементу по символьному индексу:

```
L[s]++;
```

а для вставки элемента в массив используется метод **insert** (от англ. *insert* - вставка):

```
L.\textbf{insert}(pair<string,int>(s,1));
```

В качестве аргумента этому методу передается пара «строка-целое» (**pair<string,int>**), составленная из слова **s** и числа 1 (первое вхождение этого слова, счетчик равен 1).

Таким образом, основной цикл получается очень простой:

```
while (Fin>>s )
{
int p;
p=L.count (s);
if (p==1) L[s]++;
else
L.insert (pair <string,int> (s,1));
}
```

Здесь **Fin**-это входной файловый поток , а **s**-переменная типа **string**. Запись **while(Fin>>s)** обозначает «пока чтение очередного слова из файла завершается удачно».

Более того, это цикл можно еще упростить так:

```
while (Fin>>s) L[s]++;
```

Дело в том, что при обращении к неизвестному элементу в контейнере **map** он будет создан автоматически. Это значит, что если прочитанного слова еще не было в словаре, создается новый элемент, в него записывается это слово и счетчик устанавливается в нуль. Затем мы сразу увеличиваем счетчик до 1 с помощью оператора ++.

Осталась одна нерешенная задача: вывести результаты в файл. Контейнер **map** позволят легко найти элемент по символьному индексу, но как перебрать *все* элементы, добавленные в контейнер?

Для этого служат итераторы (или курсоры) - специальные объекты, которые позволяют перебрать все элементы контейнера. Итератор указывает на текущий объект в ассоциативном массиве, с которым мы можем что-то сделать, например, вывести его данные в файл[11].

Итератор для нашего контейнера нужно объявить так:

```
map<string,int>::iterator it;
```

Этот итератор предназначен для контейнеров, состоящих из пар «строка-целое», к которым относится и наш контейнер **L**. Для того, чтобы установить итератор на первый по счету элемент контейнера, напишем

```
it=L.begin();
```

Итератор - это специальный указатель на элемент. Поэтому обращаться к полям этого элемента нужно так же, как к полям структуры по указателю, с помощью оператора ->. У элемента контейнера **map** два поля, первое (с именем **first**)-строка, второе (с именем **second**)-целое число. Поэтому вывести в выходной поток **Fout** данные по очередному элементу можно так:

```
Fout<<it->first<<":"<<it->second;
```

Оператор ++ сдвигает итератор к следующему элементу. Если значение итератора после очередного продвижения совпало с **L.end()**, значит все элементы коллекции уже просмотрены. Таким образом, цикл, который выводит данные по всем словам из словаря, выглядит так:

```
for (it=L.begin(); it!=L.end(); it++)
Fout<<it->first<<":"<<it->second<<endl;
```

Теперь можно записать всю программу целиком:

```
#include <iostream>
#include <fstream>
#include <map>
using namespace std;
main ()
{
ifstream Fin;
ofstream Fout;
string s;
map <string,int> L;
Fin.open ("input.txt");
while (Fin>>s)
L[s]++;
Fin.close();
Fout<<it->first<<":"<<it->second<<endl;
Fout.close();
}
```

1.21 Стек, очередь, дек

1.21.1 Использование контейнера stack

Стеком принято называть линейные списки, элменеты которых добавляются, а так же, удаляются только с одного конца [11].

Задача. *Дан файл, в котором находятся целые числа. Необходимо вывести эти числа в другой файл, но при этом они должны быть записаны в обратном порядке.*

Именно в таких задачах принято использовать стек:

— необходимо добавить элемент на вершину стека (англ. *push* - втолкнуть);

— получить элемент с вершины стека и удалить его из стека (англ. *pop* - вытолкнуть).

Посмотрим решение данной задачи в псевдокоде. Первым действием читаем наши данные и добавляем в стек:

```
пока файл не пуст
```

```
{
        прочитать x
        добавить x в стек
}
```

После такой операции самым верхним элементом останется последнее число, и нам остается только «вытолкнуть» эти числа, и они окажутся записанными в обратном порядке:

```
пока в нашем стеке есть элементы
{
вытолкнуть элементы из этого стека в x
записать полученный x в новый файл
}
```

Библиотека STL содержит стандартный тип данных **stack**, который можно использовать для хранения элементов любого типа. Для использования стека необходимо подключить заголовочный файл **stack**. Стек для целых чисел объявляется так:

```
#include <stack>
...
stack <int> S;
```

У контейнера **stack** есть готовые методы, которые мы будем использовать:

— **push** - добавление элемента на вершину стека;

— **pop** - снятие элемента с вершины стека;

— **top** - функция, возвращающая верхний элемент стека, не удаляя его;

— **empty** - логическая функция, которая возвращает истинное значение, если стек пуст, и ложное в противном случае.

Полная программа решения задачи реверса массива выглядит так:

```
#include <iostream>
#include <fstream>
#include <stack>
using namespace std;
main ()
```

```
{
ifstream Fin;
ofstream Fout;
stack <int> S;
int x;
// чтение данных из файла
Fin.open ("input.dat");
while (Fin>>x )
S.push (x);
Fin.close();
// запись результата в файл
Fout.open ( "output.dat" );
while (! S.empty())
{
Fout<<S.top()<<endl;
S.pop();
}
Fout.close();
}
```

Программа получился коротким, так как все необходимые функции для работы со стеком уже реализованы в библиотеке STL.

1.21.2 Очередь. Дек

Если для линейного списка введены две операции, такие как добавление нового элемента в конец очереди и удаление первого элемента из очереди, тогда линейный список будет называться очередью.

Для него необходимо ввести 2 подпрограммы:

— процедура **Put** добавляющий в конец очереди новый элемент; при необходимости массив может расширятся блоками по 10 элементов;

— функция **Get** возвращающий первый элемент очереди и удаляющий его из очереди; остальные элементы будут сдвигаться к началу массива.

Мы уже знаем определение стека, дадим определение для дека.

Как и стек, **Дек** является линейным списком, но здесь элементы можно удалять и добавлять не только с одного конца, но и с другого конца.

Хорошим примером для лека является моделирование колоды игральных карт.

В составе библиотеки STL находится тип **deque**, оно предназначено для работы с деком. Это помогает быстро вставить и быстро удалить элементы на концах дека[12].

2 ГЛАВА 2. Решения задач ЕГЭ по программированию

2.1 Вычисление контрольного значения

Пример. По каналу связи передается последовательность положительных целых чисел. Все числа не превышают 1000, их количество известно, но может быть очень велико. Затем передается контрольное значение - наибольшее число R, удовлетворяющее следующим условиям

1) R - произведение двух различных переданных элементов последовательности («различные» означает, что не рассматриваются квадраты переданных чисел, произведения различных, но равных по величине элементов допускаются);

2) R не делится на 10.

В результате помех при передаче как сами числа, так и контрольное значение могут быть искажены.

Напишите программу которая будет проверять правильность контрольного значения. Программа должна напечатать отчет по следующей форме:

Получено чисел: …

Принятое контрольное значение: …

Вычисленное контрольное значение: …

Контроль пройден (или Контроль не пройден)

Если удовлетворяющее условию контрольное значение определить невозможно, вычисленное контрольное значение не выводится, но выводится фраза «Контроль не пройден».

Перед текстом программы кратко опишите алгоритм решения и укажите язык программирования и его версию.

Вам предлагаются два задания, связанные с этой задачей: задание А и задание Б. Вы можете решать оба задания А и Б или одно из них по своему выбору.

Итоговая оценка выставляется как максимальная из оценок за задания А и Б. Если решение одного из заданий не представлено, то считается, что оценка за это задание составляет 0 баллов.

Задание Б является усложненным вариантом задания А, оно содержит дополнительные требования к программе.

А. Напишите на любом языке программирования программу для решения поставленной задачи, в которой входные данные будут запоминаться в массиве, после чего будут проверены все возможные пары элементов.

Обязательно укажите, что программа является решением **задания А.**

Максимальная оценка за выполнение задания А - 2 балла.

Б. Напишите программу для решения поставленной задачи, которая будет эффективна как по времени, так и по памяти (или хотя бы по одной из этих характеристик).

Программа считается эффективной по времени, если время работы программы пропорционально количеству элементов последовательности N, т.е. при увеличении N в k раз время работы программы должно увеличиваться не более чем в k раз. **Обязательно** укажите, что программа является решением **задания Б**.

Входные данные

В первой строке указывается количество чисел N. В каждой из последующих.

N строк записано одно натуральное число, не превышающее 1000.

В последней строке записано контрольное значение.

Пример входных данных:

5
60
7
8
15
20
105

Выходные данные

Программа должна напечатать отчет по образцу, приведенному в условии.

Пример выходных данных для приведенного выше примера входных данных:

Получено чисел: 5
Принятое контрольное значение: 105
Вычисленное контрольное значение: 105
Контроль пройден

Пояснение.

Произведение двух чисел делится на 10, если один из сомножителей делится на 10 (второй может быть любым), либо если ни один из сомножителей не делится на 10, но один из сомножителей делится на 2, а другой - на 5.

Чтобы получить произведение, не делящееся на 10, нужно взять два сомножителя так, чтобы эти условия не выполнялись. Чтобы добиться этого, можно разбить все элементы входной последовательности на 4 непересека-

ющихся класса чисел:

— кратные 10 (класс 10);

— кратные 2, но не кратные 5 (класс 2);

— кратные 5, но не кратные 2 (класс 5);

— не кратные ни 2, ни 5 (класс 0).

Числа, кратные 10, можно сразу отбросить: они не могут участвовать в итоговом произведении. Произведение двух чисел не будет делиться на 10, если оба числа принадлежат одному классу, либо если числа принадлежат разным классам, но не классам 2 и 5. При этом для получения максимального значения следует брать максимально возможное число из каждого класса. Пусть a2 - максимальное число в классе 2, b2 - второе по величине число в классе 2, аналогичным образом обозначим два наибольших числа в классах 5 и 0. Тогда контрольным значением будет наибольшее из следующих произведений: a2*b2, a5*b5, a0*b0, a0*a2, a0*a5.

Программа читает все входные данные один раз, не запоминая все данные в массиве, для каждого входного числа определяет его класс, отбрасывает числа класса 10 и хранит два наибольших числа для каждого из остальных классов. После ввода всей последовательности программа вычисляет 5 перечисленных выше произведений, выбирает из них наибольшее и сравнивает его с введенным контрольным значением.[18]

```
#include <iostream>
#include <string>
using namespace std;
int main(int argc, const char * argv[]) {
#include <stdio.h>
void main();
{
int N; /*количество чисел на входе*/
int x; /*исходные данные*/
int a2=0, b2=0; /*макс. числа, кратные 2, но не кратные 5*/
int a5=0, b5=0; /*макс. числа, кратные 5, но не кратные 2*/
int a0=0, b0=0; /*максимальные числа, не кратные 5 и 2*/
int R; /*введенное контрольное значение*/
int m; /*вычисленное контрольное значение*/
int i;
cin>>N;
for (i=1; i<=N; ++i) {
```

```
cin>>x;
if (x%10==0) continue; /*ничего не делать*/
if (x%2==0) {
if (x>a2) { b2=a2; a2=x;}
else if (x>b2) b2=x;
}
else if (x%5==0) {
if (x>a5) {b5=a5; a5=x;}
else if (x>b5) b5=x;
}
else {
if (x>a0) { b0=a0; a0=x;}
else if (x>b0) b0=x;
}
}
cin>>R;
m=a0*a2;
if (a0*a5>m) m=a0*a5;
if (a0*b0>m) m=a0*b0;
if (a2*b2>m) m=a2*b2;
if (a5*b5>m) m=a5*b5;
printf("Получено␣чисел:␣%d/n", N);
printf("Принятое␣контрольное␣значение:␣%d\n", R);
if (m>0) printf("Вычисленное␣контрольное␣значение:␣%d\n", m);
if (R>40 && R==m) cout<< "Контроль␣пройден\n";
else cout<< "Контроль␣не␣пройден\n";
}
system("pause");
return 0;
}
```

2.2 Поиск основного подмножества экспериментальных значений

Пример 1. На ускорителе для большого числа частиц производятся замеры скорости каждой из них. Скорость частицы - это целое число (положительное, отрицательное или 0). Частиц, скорость которых измерена, может быть очень много, но не может быть меньше трех. Скорости всех частиц различны. При обработке результатов в каждой серии эксперимента отбирается

основное множество скоростей. Это такое непустое множество скоростей частиц (в него могут войти как скорость одной частицы, так и скорости всех частиц серии), для которого произведение скоростей является максимальным среди всех возможных множеств. При нахождении произведения знак числа учитывается. Если есть несколько таких множеств, то основным считается то, которое содержит наибольшее количество элементов.

Вам предлагается написать программу, которая будет обрабатывать результаты эксперимента, находя основное множество. Перед текстом программы кратко опишите используемый Вами алгоритм решения задачи.

На вход программе в первой строке подается количество частиц N. В каждой из последующих N строк записано одно целое число, по абсолютной величине не превышающее 10^9.

Вам предлагается два задания, связанных с этой задачей: задание А и задание Б. Вы можете решать оба задания или одно из них по своему выбору. Итоговая оценка выставляется как максимальная из оценок за задания А и Б. Если решение одного из заданий не представлено, то считается, что оценка за это задание - 0 баллов.

Задание Б является усложненным вариантом задания А, оно содержит дополнительные требования к программе.

А. Напишите на любом языке программирования программу для решения поставленной задачи, в которой входные данные будут запоминаться в массиве. Перед программой укажите версию языка программирования.

Обязательно укажите, что программа является решением задания А. Максимальная оценка за выполнение задания А — 2 балла.

Б. Напишите программу для решения поставленной задачи, которая будет эффективна как по времени, так и по памяти (или хотя бы по одной из этих характеристик). Программа считается эффективной по времени, если время работы программы пропорционально количеству полученных показаний прибора N, т.е. при увеличении N в k раз время работы программы должно увеличиваться не более чем в k раз. Программа считается эффективной по памяти, если размер памяти, использованной в программе для хранения данных, не зависит от числа N и не превышает 1 килобайта.

Перед программой укажите версию языка программирования и кратко опишите использованный алгоритм.

Обязательно укажите, что программа является решением **задания Б**. Максимальная оценка за правильную программу, эффективную по времени и по памяти, - 4 балла.

Максимальная оценка за правильную программу, эффективную по времени, но неэффективную по памяти, - 3 балла.

Напоминаем! Не забудьте указать, к какому заданию относится каждая из представленных Вами программ.

Пример входных данных:

5
123
2
-1000
0
10

Программа должна вывести в порядке возрастания номера частиц, скорости которых принадлежат основному множеству данной серии. Нумерация частиц ведется с единицы.

Пример выходных данных для приведенного выше примера входных данных:

1 2 5

Пояснение.

Основное множество состоит из всех значений скоростей, кроме 0, если он встречается, и кроме минимальной по модулю отрицательной скорости, если отрицательных скоростей нечетное число.

Программа читает все входные данные один раз, не запоминая все входные данные в массиве, размер которого равен N. Во время чтения данных запоминается номер 0, если он встретится (по условию все значения различны, поэтому 0 встречается не больше одного раза), подсчитывается количество отрицательных значений и ищется минимальное по модулю отрицательное значение. После окончания ввода распечатываются все номера, кроме номера 0 номера минимального по модулю отрицательного значения, но только в случае, если их нечетное число.

```
#include <iostream>
using namespace std;
int main() {
    int N=0;
    cin>>N;
long min=-1000000001;
int j=0, k=0, c=0;
    for (int i=1; i<=N; ++i)
```

```
{
int num;
      cin >>num;
      if (num==0)
      j=i;
      if (num<0)
      {
c+=1;
if (min<num)
      {
      min=num;
      k=i;
      }
      }
}
      for (int i=1; i<=N; ++i)
      {
      if (i!=j && (c%2==0 || i!=k))
      cout<<i<<' ';
      }
      system("pause");
      return 0;
}
```

Пример 2.

Для заданной последовательности целых неотрицательных чисел необходимо найти минимальное произведение двух ее элементов, различающихся порядковыми номерами не менее чем на 6. Значение каждого элемента последовательности не превышает 1000. Количество элементов последовательности не превышает 10000 и не менее 7.

Вам предлагается два задания, связанных с этой задачей: задание А и задание Б. Вы можете решать оба задания или одно из них по своему выбору. Итоговая оценка выставляется как максимальная из оценок за задания А и Б. Если решение одного из заданий не представлено, то считается, что оценка за это задание - 0 баллов.

Задание Б является усложненным вариантом задания А, оно содержит дополнительные требования к программе.

А. Напишите на любом языке программирования программу для решения поставленной задачи, в которой входные данные будут запоминаться в массиве, после чего будут проверены все возможные пары элементов. Перед программой укажите версию языка программирования.

Обязательно укажите, что программа является решением **задания А**. Максимальная оценка за выполнение задания А - 2 балла.

Перед программой укажите версию языка программирования и кратко опишите использованный алгоритм. **Обязательно** укажите, что программа является решением **задания Б**. Максимальная оценка за правильную программу, эффективную по времени и по памяти, - 4 балла.

Максимальная оценка за правильную программу, эффективную по времени, но неэффективную по памяти, - 3 балла.

Напоминаем! Не забудьте указать, к какому заданию относится каждая из представленных Вами программ.

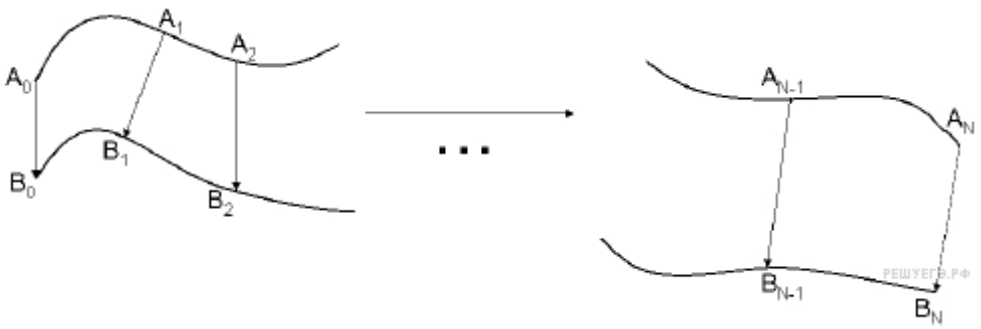
Первое число подаваемое на вход программы - количество элементов последовательности.

Решение

```
#include <iostream>
using namespace std;
int main() {
    int a[10000],min;
int i, j, N;
    int minb;
    cin>>N;
    for (i=0; i<N; i++) // ввод значений в массив
    cin>>a[i];
    min=1000*1000+1;
    for (i=0; i<N-6; i++) {
    for (j=i+6; j<N;j++){//просмотрим каждую возможность
    if (a[i] * a[j]<minb)
    minb=a[i]*a[j];
    }
    }
    cout<<minb;
    system("pause");
    return(0);
```

2.3 Анализ пар значений

Пример 1.



Гоночная трасса состоит из двух основных дорог и нескольких переездов, позволяющих перейти с одной дороги на другую. На всех участках, включая переезды, движение разрешено только в одну сторону, поэтому переезд возможен только с дороги A на дорогу B. Гонщик стартует в точке A_0 и должен финишировать в точке B_N. Он знает, за какое время сможет пройти каждый участок пути по каждой дороге, то есть время прохождения участков A_0A_1, A_1A_2, ..., $A_{N-1}A_N$, B_0B_1, B_1B_2, ..., $B_{N-1}B_N$. Время прохождения всех переездов A_0B_0, A_1B_1, ..., A_NB_N одинаково и известно гонщику. Необходимо определить, за какое минимальное время гонщик сможет пройти трассу.

Вам предлагается два задания с похожими условиями: задание А и задание Б. Вы можете решать оба задания или одно из них по своему выбору. Задание Б более сложное, его решение оценивается выше. Итоговая оценка выставляется как максимальная из оценок за задания А и Б.

Задание А. Имеется 10 пунктов A_i и 10 пунктов A_i, время прохождения всех переездов известно. Напишите программу для решения этой задачи. В этом варианте задания оценивается только правильность программы, время работы и размер использованной памяти не имеют значения.

Максимальная оценка за правильную программу - 2 балла.

Задание Б. Имеется набор данных о пунктах A_i и B_i. Напишите программу для решения этой задачи.

Постарайтесь сделать программу эффективной по времени и используемой памяти (или хотя бы по одной из этих характеристик).

Программа считается эффективной по времени, если время работы программы пропорционально количеству пар чисел N, т. е. при увеличении N в k раз время работы программы должно увеличиваться не более чем в k раз.

Программа считается эффективной по памяти, если размер памяти, использованной в программе для хранения данных, не зависит от числа N и не превышает 1 килобайта.

Максимальная оценка за правильную программу, эффективную по времени и памяти, - 4 балла.

Максимальная оценка за правильную программу, эффективную по времени, но неэффективную по памяти, - 3 балла.

Перед текстом программы кратко опишите алгоритм решения и укажите язык программирования и его версию.

Входные данные

В первой строке задается количество участков трассы N. Во второй строке задается целое число t - время (в секундах) прохождения каждого из переездов A_0B_0, A_1B_1, ..., A_NB_N. В каждой из последующих N строк записано два целых числа ai и bi, задающих время (в секундах) прохождения очередного участка на каждой из дорог. В первой из этих строк указывается время прохождения участков A_0A_1 и B_0B_1, во второй - A_1A_2 и B_1B_2 и т. д.

Пример входных данных

3

20

320 150

200 440

300 210

Выходные данные

Программа должна напечатать одно целое число: минимально возможное время прохождения трассы (в секундах).

Пример выходных данных для приведенного выше примера входных данных

750

Пояснение.

Пусть X_i - минимально возможное время движения от A_0 до A_i, а Y_i - минимально возможное время движения от A_0 до B_i. В точку A_i можно попасть только из точки A_{i-1}, поэтому $X_i = X_{i-1} + a_i$. В точку B_i можно попасть двумя способами: из точки B_{i-1} и из точки A_i. Чтобы найти минимальное время, нужно вычислить время движения каждым из этих способов и выбрать из них минимальное. Получается, что $Y_i = \min(Y_{i-1} + b_i, X_i + t)$. Кроме того, очевидно, что $X_0=0$, $Y_0=t$. Используя эти соотношения, можно найти значения X_i и Y_i для всех i от 1 до N, не сохраняя всех значений asub>i и bsub>i. Значение Y_N будет окончательным ответом задачи.

Решение

```
#include <iostream>
```

```
#include <string>
using namespace std;
int main(int argc, const char * argv[]) {
#include <stdio.h>
    void main();
    {
    int N; /* количество участков */
    int t; /* время переезда */
    int a, b; /* время прохождения участков */
    int x; /* текущее значение X */
    int y; /* текущее значение Y */
    int y1, y2;
    int i;
    cin>>N;
    cin>>t;
    x=0; y=t;
    for (i=1; i<=N; ++i) {
    cin>>a>>b;
    x=x+a;
    y1=y+b;
    y2=x+t;
    if (y1<y2) y=y1;
    else y=y2;
    }
    printf("%d\n", y);
    system("pause");
    return 0;
    }
}
```

Пример 2. Автомобиль, участвующий в гонке, может быть оснащен двумя разными типами колес (A и B). Вдоль трассы расположены станции, на которых можно выполнить замену колес A на B, эта операция занимает *t* секунд. Замена колес B на A в ходе гонки технически невозможна. На старт можно выйти с любым комплектом. Для каждого участка между станциями известно, за какое время можно пройти этот участок с каждым из комплектов колес. Необходимо определить, за какое минимальное время можно пройти всю трассу.

Напишите программу для решения этой задачи.

Перед текстом программы кратко опишите алгоритм решения и укажите язык программирования и его версию.

Вам предлагается два задания с похожими условиями: задание А и задание Б. Вы можете решать оба задания или одно из них по своему выбору. Задание Б более сложное, его решение оценивается выше. Итоговая оценка выставляется как максимальная из оценок за задания А и Б.

Задание А. Имеются данные о времени прохождения участков трассы с различными типами колес. Всего пунктов 10 штук. Напишите программу для решения этой задачи. В этом варианте задания оценивается только правильность программы, время работы и размер использованной памяти не имеют значения.

Максимальная оценка за правильную программу - 2 балла.

Задание Б. Имеются данные о времени прохождения участков трассы с различными типами колес. Пунктов может быть очень много. Постарайтесь сделать программу эффективной по времени и используемой памяти (или хотя бы по одной из этих характеристик).

Программа считается эффективной по времени, если время работы программы пропорционально количеству пар чисел N, т. е. при увеличении N в k раз время работы программы должно увеличиваться не более чем в k раз.

Программа считается эффективной по памяти, если размер памяти, использованной в программе для хранения данных, не зависит от числа N и не превышает 1 килобайта.

Максимальная оценка за правильную программу, эффективную по времени и памяти, - 4 балла.

Максимальная оценка за правильную программу, эффективную по времени, но неэффективную по памяти, - 3 балла.

Входные данные

В первой строке задается количество участков трассы N. Во второй строке задается целое число t - время (в секундах) на замену колес A на B. В каждой из последующих N строк записано два целых числа a_i и b_i, задающих время (в секундах) прохождения очередного участка с каждым из комплектов. В первой из этих строк указывается время прохождения участка от старта до первой станции, во второй - от первой станции до второй и т. д.

Пример входных данных

3

10

130 210

320 140

100 120

Выходные данные

Программа должна напечатать одно целое число: минимально возможное время прохождения трассы (в секундах).

Пример выходных данных для приведенного выше примера входных данных

400

Пояснение.

Пусть P_0 - точка старта, P_1 - первая станция, P_2 - вторая, ..., P_N - финиш. Пусть X_i - минимально возможное время от старта до отправления из P_i, при условии, что автомобиль отправляется из P_i с колесами A, а Y_i - минимально возможное время от старта до отправления из P_i, при условии, что автомобиль отправляется из P_i с колесами B. С колесами A можно выехать из P_i только при условии, что с этими же колесами автомобиль прибыл на этот пункт (установить колеса A в ходе гонки нельзя), поэтому $X_i = X_{i-1} + a_i$. Если автомобиль выехал из P_i с колесами B, они могли быть установлены либо на станции в этом пункте, либо ранее. Для нахождения Y_i нужно вычислить время для каждого из этих случаев и выбрать из них меньшее. Получается, что $Y_i = min(X_i + t, Y_{i-1} + b_i)$. Кроме того, очевидно, что $X_0 = 0$, $Y_0 = 0$.

Используя эти соотношения, можно найти значения X_i и Y_i для всех i от 1 до N, помня лишь текущие значения a_i и b_i. Окончательным ответом задачи будет меньшее из значений X_N и Y_N.

Решение

```
#include <iostream>
#include <string>
#include <stdio.h>
using namespace std;
int main(int argc, const char * argv[]) {
void main();
{
int N; /* количество участков */
int t; /* время замены */
int a, b; /* время прохождения участков */
int x; /* текущее значение X */
int y; /* текущее значение Y */
```

```
int y1, y2;
int i;
cin>>N;
cin>>t;
x=0; y=0;
for (i=1; i<=N; ++i) {
cin>>a>>b;
x=x+a;
y1=x+t;
y2=y+b;
if (y1<y2) y=y1;
else y=y2;
}
if (x<y) cout<<x;
else cout<<y;
}
system("pause");
return 0;
}
```

Пример 3. На вход программы поступает последовательность из N целых положительных чисел, все числа в последовательности различны. Рассматриваются все пары различных элементов последовательности, находящихся на расстоянии не меньше чем 4 (разница в индексах элементов пары должна быть 4 или более, порядок элементов в паре неважен). Необходимо определить количество таких пар, для которых произведение элементов делится на 29.

Описание входных и выходных данных

В первой строке входных данных задается количество чисел N ($4 \le N \le 1000$). В каждой из последующих N строк записано одно целое положительное число, не превышающее 10 000.

В качестве результата программа должна вывести одно число: количество пар элементов, находящихся в последовательности на расстоянии не меньше чем 4, в которых произведение элементов кратно 29.

Пример входных данных:

```
7
58
2
```

3

5

4

1

29

Пример выходных данных для приведенного выше примера входных данных:

5

Пояснение. Из 7 заданных элементов с учетом допустимых расстояний между ними можно составить 6 произведений: 58*4, 58*1, 58*29, 2*1, 2*29, 3*29. Из них на 29 делятся 5 произведений.

Требуется написать эффективную по времени и по памяти программу для решения описанной задачи.

Программа считается эффективной по времени, если при увеличении количества исходных чисел N в k раз время работы программы увеличивается не более чем в k раз.

Программа считается эффективной по памяти, если память, необходимая для хранения всех переменных программы, не превышает 1 Кбайт и не увеличивается с ростом N.

Максимальная оценка за правильную (не содержащую синтаксических ошибок и дающую правильный ответ при любых допустимых входных данных) программу, эффективную по времени и по памяти, - 4 балла.

Максимальная оценка за правильную программу, эффективную только по времени - 3 балла.

Максимальная оценка за правильную программу, не удовлетворяющую требованиям эффективности, - 2 балла.

Вы можете сдать одну программу или две программы решения задачи (например, одна из программ может быть менее эффективна). Если Вы сдадите две программы, то каждая из них будет оцениваться независимо от другой, итоговой станет бо?льшая из двух оценок.

Перед текстом программы обязательно кратко опишите алгоритм решения. Укажите использованный язык программирования и его версию.

Пояснение.

Произведение двух чисел делится на 29, если хотя бы один из сомножителей делится на 29.

При вводе чисел можно подсчитывать количество чисел, кратных 29, не считая четырех последних. Обозначим их *n29*.

Сами числа, кроме четырех последних, при этом можно не хранить.

Очередное считанное число будем рассматривать как возможный правый элемент искомой пары. Если очередное считанное число делится на 29, то к ответу следует прибавить количество чисел до него, не считая четырех последних (включая считанное). Если очередное считанное число на 29 не делится, то к ответу следует прибавить *n29*.

Чтобы построить программу, эффективную по памяти, заметим, что, поскольку при обработке очередного элемента входных данных используются значения, находящиеся на четыре элемента ранее, достаточно хранить только четыре последних элемента или информацию о них.

Решение.

```
#include <iostream>
using namespace std;
int main()
{
    int s=4; //требуемое расстояние между элементами
    int n;
    int n1=0, n2=0, n3=0, n4=0;
    //хранение последних s счетчиков
    int a_; // очередное значение
    int cnt; // количество искомых пар
    cin>>n;
    cnt=0;
    for (int i=0; i<n; ++i)
    {
    cin>>a_; // считано очередное значение
    if (i>=s)
    {
    if (a_%29==0)
    cnt+=i-s+1;
    else
    cnt+=n4;
    }
    //сдвигаем элементы счетчиков
    n4=n3;
    n3=n2;
    n2=n1;
```

```
        //обновляем счетчик кратных 29
        if (a_%29==0)
        n1+=1;
        }
        cout<<cnt;
        system("pause");
        return 0;
}
```

ГЛАВА 3. Задачи для самостоятельного решения

Практическая работа 1

Простые вычисления

Уровень А. Написать программу, которая вычисляет сумму, произведение и среднее арифметическое трех целых чисел. Пример:

Введите три целых числа:

2 4 9

2+4+9=15

2*4*9=72

(2+4+9)/3=5

Уровень В. Написать программу, которая вычисляет длину отрезка XY, где X и Y координаты двух точек на плоскости. Пример:

Введите координаты точки X:

1 3

Введите координаты точки B:

1 1

Длина отрезка AB = 2

Уровень С. Напишите программу, которая вводит через запятую отдельные цифры введенного трехзначного числа. Пример:

Получено число 789.

Его цифры 7, 8, 9.

Практическая работа 2

Ветвления

Уровень А. Напишите программу, которая находит максимальное число из введенных трех целых чисел. Пример:

Введите три целых числа:

4 8 3

Максимальное число 8

Уровень В. Напишите программу, которая находит максимальное число из введенных пяти целых чисел. Пример:

Введите пять целых чисел:

2 4 6 7 5

Максимальное число 7

Уровень С. Напишите программу, которая определяет кто старше (Влад, Максим и Сергей) по введенным годам. Пример:

Возраст Влада: 21

Возраст Максима: 28

Возраст Сергея: 8

Ответ: Максим старше всех.

Пример:

Возраст Влада: 19

Возраст Максима: 19

Возраст Сергея: 22

Ответ: Влад и Максима младше Сергея.

Практическая работа 3

Сложные условия

Уровень А. Ввести три числа, вывести количество одинаковых чисел в этой цепочке. Пример:

Введите три числа:

9 9 9

Все числа одинаковые.

Пример:

Введите три числа:

2 4 2

Два числа одинаковые.

Пример:

Введите три числа:

3 5 9

Нет одинаковых чисел.

Уровень В. Напишите программу, которая по соответственному номеру месяца и выводит время года или сообщение об ошибке. Пример:

Введите номер месяца:

6

Июнь.

Пример:

Введите номер месяца:

20

Неверный номер месяца.

Уровень С. Напишите программу, получающая возраст человека (целое число, не превышающее 99) и выводит этот возраст со словом «год», «года» или «лет». Например, «1 год», «44 года», ’<18 лет». Пример:

Введите возраст: 19

Вам 19 лет.

Пример:

Введите возраст: 1

Вам 1 год.

Пример:

Введите возраст: 44

Вам 44 года.

Практическая работа 4

Циклы с условием

Уровень А. Напишите программу, получающая два целых числа A и B (0 < A < B) и которая выводит квадраты всех натуральных чисел в интервале от A до B.

Пример:

Введите два целых числа:

5 8

5*5=25

6*6=36

7*7=49

Уровень В. Напишите программу, получающая два целых числа и находящая их произведение, не используя операцию умножения. Учитывать, что числа могут быть и отрицательными.

Пример:

Введите два числа:

21 -3

21*(-3)=-63

Уровень С. Ввести натуральное число N и вычислить сумму всех чисел Фибоначчи, меньших N. Предусмотрите защиту от ввода отрицательного числа N.

Пример:

Введите число N:

10000

Сумма 17710

Циклы с условием - 2

Уровень А. Напишите программу, которая находит сумму цифр числа, введенного в компилятор.

Пример:

Введите натуральное число:

45678

Сумма цифр 30.

Уровень B. Напишите программу, которая определяет, есть ли в введенном натуральном числе, две одинаковые цифры, которые стоят рядом.

Пример:

Введите натуральное число:

2583

Нет.

Пример:

Введите натуральное число:

36687

Да.

Уровень C. Напишите программу, которая определяет, есть ли в введенном натуральном числе, две одинаковые цифры (не обязательно стоящие рядом).

Пример:

Введите натуральное число:

27932

Да.

Пример:

Введите натуральное число:

69871

Нет.

Пратическая работа 5

Циклы с переменной

Уровень A. Напишите программу, которая находит все пятизначные числа, при делении на 133 дающие в остатке 125, а при делении на 134 в остатке 111.

Уровень B. Числом Армстронга называют сумму цифр числа, возведенных в N-ную степень (где N - количество цифр в числе) равную самому числу. Например, $153 = 1^3 + 5^3 + 3^3$. Напишите программу, которая находит все трехзначные Армстронга.

textitУровень C. Автоморфным числом называют число, которое равно последним цифрам своего квадрата. Например, $35^2 = 1225$. Напишите программу, которая получает натуральное число N и выводит на экран все автоморфные числа, не превосходящие N.

Пример:

Введите N:

1000

1*1=1

5*5=25

6*6=36

25*25=625

76*76=5776

Вложенные циклы

Уровень А. Напишите программу, получающая натуральные числа A и B (A<B) и которая выводит все простые числа в интервале от A до B.

Пример:

Введите границы диапазона:

0 9

1 2 3 5 7

Уровень В. В магазине продается мастика в ящиках по 16 кг, 18 кг, 22 кг. Как купить ровно 188 кг мастики, не вскрывая ящики? Сколькими способами можно это сделать?

Уровень С. Напишите программу, которая при введении натурального числа N выводит все натуральные числа, не превосходящие N и делящиеся на каждую из своих цифр.

Пример:

Введите N:

15

12 3 4 5 6 7 8 9 11 12 15

Пратическая работа 6

Процедуры

Уровень А. Напишите процедуру, принимающая параметр - натуральное число N – и выводит на экран линию из N символов ’-’.

Пример:

Введите N:

9

- - - - - - - - -

Уровень В. Напишите процедуру, выводящая на экран в столбик все цифры переданного ей числа, начиная с первой.

Пример:

Введите натуральное число:

5678

5

6

7

8

Уровень С. Напишите процедуру, выводящая на экран запись переданного ей числа в римской системе счисления.

Пример:

Введите натуральное число:

2013

MMXIII

Процедуры с изменяемыми параметрами

Уровень А. Напишите процедуру, переставляющая три переданные ей числа в порядке возрастания.

Пример:

Введите три натуральных числа:

9 21 3

3 9 21

Уровень В. Напишите процедуру, сокращающая дробь вида M/N. Числитель и знаменатель дроби передаются как изменяемые параметры.

Пример:

Введите числитель и знаменатель дроби:

35 49

После сокращения: 5/7

Уровень С. Напишите процедуру, вычисляющая наибольший общий делитель и наименьшее общее кратное двух натуральных чисел и возвращающая их через изменяемые параметры.

Пример:

Введите два натуральных числа:

10 15

НОД(10,15)=5

НОК(10,15)=30

Практическая работа 7

Функции

Уровень А. Напишите функцию, определяющую сумму цифр переданного ей числа.

Пример:

Введите натуральное число:

456

Сумма цифр числа 456 равна 15.

Уровень В. Напишите функцию, находящего наибольший общий делитель двух натуральных чисел.

Пример:

Введите два натуральных числа:

7006652 112307574

НОД(7006652,112307574) = 1234.

Уровень С. Напишите функцию, которая «переворачивает» число, или возвращает число, в котором цифры стоят в обратном порядке.

Пример:

Введите натуральное число:

6789

После переворота: 9876.

Логические функции

Уровень А. Напишите логическую функцию, определяющая, является ли переданное ей число совершенным, то есть, равно ли оно сумме своих делителей, меньших его самого.

Пример:

Введите натуральное число:

28

Число 28 совершенное.

Пример:

Введите натуральное число:

29

Число 29 не совершенное.

Уровень В. Напишите логическую функцию, определяющая, являются ли два переданные ей числа взаимно простыми, т.е., не имеющими общих делителей, кроме 1.

Пример:

Введите два натуральных числа:

19 39

Числа 19 и 39 взаимно простые.

Пример:

Введите два натуральных числа:

106 214

Числа 106 и 214 не взаимно простые.

*Уровень С.*Гиперпростое число - это любое число, в котором получающееся числа откидыванием нескольких цифр, тоже является простым. На-

пример, число 733 - гиперпростое, так как и оно само, и числа 73 и 7 - простые. Напишите логическую функцию, которая определяет, верно ли, что переданное ей число - гиперпростое.

Пример:

Введите натуральное число:

733

Число 733 гиперпростое.

Пример:

Введите натуральное число:

19

Число 19 не гиперпростое.

Практическая работа 9

Рекурсия

Уровень А. Напишите рекурсивную функцию, вычисляющая НОД двух натуральных чисел, используя модифицированный алгоритм Евклида.

Пример:

Введите два натуральных числа:

7006652 112307574

НОД(7006652,112307574) = 1234.

Уровень В. Напишите рекурсивную функцию, раскладывающая число на простые сомножители.

Пример:

Введите натуральное число:

378

378 = 2*3*3*3*7

Уровень С.Напишите программу получающая и выводящая на экран все возможные различные способы представления этого числа (число N) в виде суммы натуральных чисел (то есть, 5 + 6 и 6 + 5 - это один и тот же способ разложения числа 3). Решите задачу с помощью рекурсивной процедуры.

Пример:

Введите натуральное число:

3

1 + 1 + 1

1 + 2

Практическая работа 10

Перебор элементов массива

Уровень А. Напишите программу, в котором после заполнения массива

случайными числами в интервале [0,100] находит среднее арифметическое его значений.

Пример:

Массив:

6 7 8 9 10

Среднее арифметическое 8.0000

Уровень B. Напишите программу, в котором после заполнения массива случайными числами в интервале [0,100] подсчитывается отдельно среднее значение всех элементов, которые <50, и среднее значение всех элементов, которые больше или равно 50.

Пример:

Массив:

4 1 51 4 61

Среднее арифметическое элементов [0,50): 3.000

Среднее арифметическое элементов [50,100): 56.000

Уровень C. Напишите программу, в котором массив заполняется случайными числами в интервале [1,N] так, чтобы в массив обязательно входят все числа от 1 до N (постройте случайную перестановку).

Линейный поиск

Уровень A. Напишите программу, массив в котором заполняется случайными числами в интервале [0,5], вводится число X и находятся все значения, равные X.

Пример:

Массив:

13 15 13 34 5

Что ищем:

13

Нашли: A[1]=13, A[3]=13

Пример:

Массив:

5 7 9 34 6

Что ищем:

15

Ничего не нашли.

Уровень B. Напишите программу, массив в котором заполняется случайными числами в интервале [0,5] и определяется, есть ли в нем элементы с одинаковыми значениями, стоящие рядом.

Пример:

Массив:

5 6 8 8 9

Есть: 8

Уровень С. Напишите программу, массив в котором заполняется случайными числами и определяется, есть ли в нем элементы с одинаковыми значениями, не обязательно стоящие рядом.

Пример:

Массив:

8 9 5 3 4 8 9

Есть: 8, 9

Пример:

Массив:

5 8 9 6 3

Нет

Поиск максимального элемента массива

Уровень А. Напишите программу, массив в котором заполняется случайными числами и находятся минимальный и максимальный элементы массива и их номера.

Пример:

Массив:

7 8 9 6 3 7

Минимальный элемент: A[1]=3

Максимальный элемент: A[5]=9

Уровень В. Напишите программу, массив в котором заполняется случайными числами и находятся два максимальных элемента массива и их номера.

Пример:

Массив:

5 5 3 4 1

Максимальный элемент: A[1]=5

Второй максимум: A[2]=5

Уровень С. Напишите программу, массив в котором заполняется с клавиатуры и найти (за один проход) количество элементов, имеющие максимальное значение.

Пример:

Массив:

7 8 8 8 9 9 7 7

Максимальное значение 3

Количество элементов 3

Алгоритмы обработки массивов

Уровень А. Напишите программу, массив в котором заполняется случайными числами и выполняется циклический сдвиг элементов массива вправо на 1 элемент.

Пример:

Массив:

5 6 7 8 9

Результат:

9 5 6 7 8

Уровень В. Напишите программу, в котором массив имеет четное число элементов. Заполните массив случайными числами и выполните реверс отдельно в первой половине и второй половине.

Пример:

Массив:

4 5 6 7 8 9

Результат:

6 5 4 9 8 7

Уровень С. Напишите программу, в котором массив заполняется случайными числами в интервале [-100,100] и переставляется элементы так, чтобы все положительные элементы стояли в начала массива, а все отрицательные и нули - в конце. Вычислите количество положительных элементов.

Пример:

Массив:

20 -90 15 -34 10 0

Результат:

20 15 10 -90 -34 0

Количество положительных элементов: 3

Практическая работа 11

Отбор элементов массива по условию

Уровень А. Напишите программу, в котором массив заполняется случайными числами в интервале [-10,10] и отбирающий в другой массив все четные отрицательные числа.

Пример:

Массив А:

-5 6 7 -4 -6 8 -8

Массив B:

-4 -6 -8

Уровень B. Напишите программу, в котором массив заполняется случайными числами в интервале [0,100] и отбирающий в другой массив все простые числа. Используйте логическую функцию, определяющая, является ли переданное ей число простым.

Пример:

Массив A:

12 13 85 96 47

Массив B:

13 47

Уровень C. Напишите программу, в котором массив заполняется случайными числами и отбирающий в другой массив все числа Фибоначчи. Используйте логическую функцию, определяющая, является ли переданное ей число числом Фибоначчи.

Пример:

Массив A:

12 13 85 34 47

Массив B:

13 4

Практическая работа 12

Сортировка. Метод выбора

Уровень A. Известно, что массив содержит четное количество элементов. Напишите программу, сортирующая первую половину массива по возрастанию, а вторую - по убыванию. Каждый элемент должен остаться в «своей» половине.

Пример:

Массив:

5 3 4 2 1 6 3 2

После сортировки:

2 3 4 5 6 3 2 1

Уровень B. Напишите программу, сортирующая массив и находящая количество различных чисел в нем.

Пример:

Массив:

5 3 4 2 1 6 3 2 4

После сортировки:

1 2 2 3 3 4 4 5 6

Различных чисел: 6

Уровень С. Напишите программу, сравнивающая число перестановок элементов при использовании сортировки «пузырьком» и методом выбора. Проверить на разных массивах, содержащих 1000 случайных элементов, вычислить среднее число перестановок для каждого метода.

Сортировка. Метод пузырька

Уровень А. Напишите программу, в которой сортировка выполняется «методом камня» - самый «тяжелый» элемент опускается в конец массива.

Уровень В. Напишите вариант метода пузырька, который заканчивает работу, если на очередном шаге внешнего цикла не было перестановок.

Уровень С. Напишите программу, которая сортирует массив по убыванию суммы цифр числа. Используйте функцию, которая определяет сумму цифр числа.

Практическая работа 13

Посимвольная обработка строк

Уровень А. Напишите программу, в котором вводится с клавиатуры символьная строку и заменяется в ней все буквы «а» на «б» и все буквы «б» на «а» (заглавные на заглавные, строчные на строчные).

Пример:

Введите строку:

ааббААББссСС

Результат:

ббааББААссСС

Уровень В. Написать программу, в котором определяется сколько в символьной строке слов. Словом считается последовательности непробельных символов, отделенная с двух сторон пробелами (или стоящая с краю строки). Слова могут быть разделены несколькими пробелами, в начале и в конце строки тоже могут быть пробелы.

Пример:

Введите строку:

Я вышел за хлебом.

Найдено слов: 4

Уровень С. Напишите программу, в котором вводится символьная строка и находится самое длинное слово и его длина. Словом считается последовательности непробельных символов, отделенная с двух сторон пробелами (или стоящая с краю строки). Слова могут быть разделены несколькими про-

белами, в начале и в конце строки тоже могут быть пробелы.

Пример:

Введите строку:

Я вышел за хлебом.

Самое длинное слово: хлебом, длина 6

Практическая работа 14

Функции для работы со строками

Уровень А. Напишите программу, выводящая фамилию и инициалы с введенных в одну строку фамилии, имени и отчество, разделенные пробелом.

Пример:

Введите фамилию, имя и отчество:

Васильев Сергей Павлович

С.П. Васильев

Уровень В. Напишите программу, в котором вводится адрес файла и «разбирается» на части, разделенные знаком ’/’. Каждая часть выводится в отдельной строке.

Пример:

Введите адрес файла:

C:/Фото/2019/Поход/vasya.jpg

C:

Фото

2019

Поход

vasya.jpg

Уровень С. Напишите программу, заменяющая во всей строке одну последовательность символов на другую.

Пример:

Введите строку:

(X > 0) and (Y < X) and (Z > Y) and (Z <> 5)

Что меняем: and

Чем заменить: &

Результат

(X > 0) & (Y < X) & (Z > Y) & (Z <> 5)

Практическая работа 15

Сравнение и сортировка строк

Уровень А. Вводится 5 строк, в которых сначала записан порядковый номер строки с точкой, а затем - слово. Напишите программу, выводящая слова в алфавитном порядке.

Пример:

Введите 5 строк:

1. машина

2. автобус

3. жизнь

4. ложка

5. рама

Список слов в алфавитном порядке:

автобус, жизнь, ложка, машина, рама.

Уровень В. Вводится несколько строк (не более 20), в которых сначала записан порядковый номер строки с точкой, а затем - слово. Ввод заканчивается пустой строкой. Напишите программу, выводящую введенные слова в алфавитном порядке.

Пример:

Введите слова:

1. рама

2. атобус

Список слов в алфавитном порядке:

автобус, рама

Уровень С. Вводится несколько строк (не более 20), в которых сначала записаны инициалы и фамилии работников фирмы. Ввод заканчивается пустой строкой. Напишите программу, сортирующую строки в алфавитном порядке по фамилии.

Пример:

Введите ФИО:

В.Д. Вафин

А.Г. Белый

Б.В. Васильев

Список в алфавитном порядке:

А.Г. Белый

Б.В. Васильев

В.Д. Вафин

Обработка символьных строк: сложные задачи

Уровень А. На вход программы подаются данные о результатах районной олимпиады. В первой строке - количество участников N, а следующие N строк имеют следующий формат:

<Фамилия> <Имя> <Баллы>

Здесь <Фамилия> - строка, состоящая не более чем из 20 символов; <Имя> - строка, состоящая не более чем из 15 символов; <Баллы> - целое число, обозначающее общее количество баллов, набранное участником. Все данные разделены пробелами. Пример входной строки:

Иванов Петя 152

На городскую олимпиаду проходят участники, набравшие в сумме более 100 баллов. Требуется вывести количество участников, прошедших на городскую олимпиаду.

Уровень В. На вход программы подаются данные о результатах районной олимпиады. В первой строке - количество участников N, а следующие N строк имеют следующий формат:

<Фамилия> <Имя> <Баллы>

Практическая работа 16

Матрицы

Уровень А. Напишите программу, заполняющая квадратную матрицу случайными числами в интервале [10,99], и находит максимальный и минимальный элементы в матрице и их индексы.

Пример:

Матрица A:

12 14 67 45

32 87 45 63

69 45 14 11

40 12 35 15

Максимальный элемент A[2,2]=87

Минимальный элемент A[3,4]=11

Уровень В. Пиксели рисунка закодированы числами от 0 до 255 (обозначающими яркость пикселей) в виде матрицы, содержащей N строк и M столбцов. Нужно преобразовать рисунок в черно-белый по следующему алгоритму:

— вычислить среднюю яркость пикселей по всему рисунку

— все пиксели, яркость которых меньше средней, сделать черными (записать код 0), а остальные - белыми (код 255)

Пример:

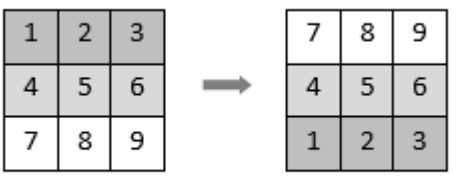
Матрица A:
12 14 67 45
32 87 45 63
69 45 14 11
40 12 35 15
Средняя яркость 37.88
Результат:
0 0 255 255
0 255 0 255
255 255 0 0
255 0 0 0

Обработка блоков матрицы

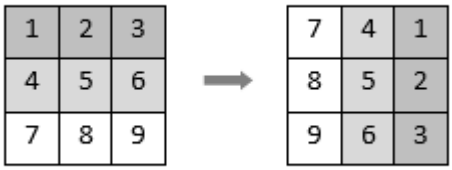
Уровень A. Напишите программу, заполняющая квадратную матрицу случайными числами в интервале [10,99], а затем записывает нули во все элементы выше главной диагонали. Алгоритм не должен изменяться при изменении размеров матрицы.

Пример:
Матрица A:
12 14 67 45
32 87 45 63
69 45 14 30
40 12 35 65
Результат:
12 0 0 0
32 87 0 0
69 45 14 0
40 12 35 65

Уровень B. Дан некий рисунок, пиксели которого закодированы числами (обозначающими цвет) в виде матрицы, содержащей N строк и M столбцов. Выполните отражение рисунка сверху вниз:



Уровень С. Дан некий рисунок, пиксели которого закодированы числами (обозначающими цвет) в виде матрицы, содержащей N строк и M столбцов. Выполните поворот рисунка вправо на 90 градусов:



Практическая работа 17

Файловый ввод и вывод

Уровень А. Напишите программу, находящая среднее арифметическое всех чисел, которые записаны в файле в столбик, и выводит результат в другой файл.

Уровень В. Напишите программу, находящая минимальное и максимальное среди четных положительных чисел, которые записаны в файле, и выводит результат в другой файл. Учтите, что таких чисел может вообще и не быть.

Уровень С. В файле в столбик записаны целые числа, сколько их - неизвестно. Напишите программу, определяющую длину самой длинной цепочки идущих подряд одинаковых чисел и выводящая результат в другой файл.

Обработка массивов из файла

Уровень А. Пусть в файле записано не более 100 чисел. Напишите программу, которая сортирует их по возрастанию последней цифры и записывает в другой файл.

Уровень В. Пусть в файле записано не более 100 чисел. Напишите программу, которая сортирует их по возрастанию последней цифры и записывает в другой файл. Используйте функцию, которая вычисляет сумму цифр числа.

Уровень С. Пусть в двух файлах записаны отсортированные по возрастанию массивы неизвестной длины. Объединить их и записать результат в третий файл. Полученный массив также должен быть отсортирован по возрастанию.

Обработка смешанных данных из файла

Уровень А. В файле записаны данные о результатах сдачи экзамена. Каждая строка содержит фамилию, имя и количество баллов, разделенные пробелами:

<Фамилия> <Имя> <Количество баллов>

Напишите программу, выводящая в другой файл фамилии и имена тех учеников, получившие больше 80 баллов.

Уровень В. В предыдущей задаче добавить к полученному списку нумерацию, сократить имя до одной буквы и поставить перед фамилией:

1) П. Иванов

2) И. Петров

3) …

Уровень С. В файле записаны данные о результатах сдачи экзамена. Каждая строка содержит фамилию, имя и количество баллов, разделенные пробелами:

<Фамилия> <Имя> <Количество баллов>

Напишите программу, выводящая в другой файл фамилии и имена тех учеников, получившие больше 80 баллов. Список должен быть отсортирован по убыванию балла. Формат выходных данных:

1) П. Иванов 98

2) И. Петров 96

3) …

Практическая работа 18

Динамические массивы

1. Введите с клавиатуры число N и вычислите все простые числа в диапазоне от 2 до N, используя решето Эратосфена.

2. Введите с клавиатуры число N и запишите в массив первые N простых чисел.

3. Введите с клавиатуры число N и запишите в массив первые N чисел Фибоначчи.

4. Напишите функцию, находящая максимальный элемент переданного ей динамического массива.

5. Напишите подпрограмму, находящая максимальный и минимальный элементы переданного ей динамического массива (используйте изменяемые параметры).

6. Напишите рекурсивную функцию, считающая сумму элементов переданного ей динамического массива.

7. Напишите функцию, сортирующая значения переданного ей динамического массива, используя алгоритм «быстрой сортировки».

Практическая работа 19

Модули

В программе, составляющая алфавитно-частотный словарь , вынесите

все операции со списком в отдельный модуль.

Список литературы

[1] Викентьева О.Л. Конспект лекций по курсу Алгоритмические языки и программирование (Основы языка С++, I семестр).- М.: Пермь: ПГТУ, 2003.– 80 с.

[2] Герберт Ш. С++: базовый курс, 3е издание.: Пер. с англ. - М.: Издательский дом «Вильямс», 2010.– 624с.

[3] Давлетшина Р.Р., Давлетшин А.И. Практикум по программированию. /Р.Давлетшина, А.Давлетшин./ М.:Казань: ТГГПУ, 2008.–64 с.

[4] Крупник А. Изучаем Си / А. Крупник. – М.: Питер 2001. – 233 с.

[5] Литвиненко Н.А. Технология программирования на С++ / Н. Литвиненко. – М.: БХВ-Петербург,Лаборатория знаний 2010. – 281 с.

[6] Наместников С.М. Основы программирования на языке С++: Учебное пособие./Наместников С.- М.: Ульяновск: УлГТУ, 2007.

[7] Пахомов Б. С/С++ и MS Visual C++ 2010 для начинающих / Б. Пахомов. – М.: БХВ-Петербург 2011.– 726 с.

[8] Павловская Т.А. С/С++. Программирование на языке высокого уровня. /Т.Павловская.- М.: Питер, 2003.–461 с.

[9] Поляков К.Ю. Еремин Е.А. Информатика. Углубленный уровень: учебник для 10 кл.В 2ч. Ч1 / К. Поляков, Е. Еремин. – М.: БИНОМ,Лаборатория знаний 2014. – 344 с.

[10] Поляков К.Ю. Еремин Е.А. Информатика. Углубленный уровень: учебник для 10 кл.В 2ч. Ч2 / К. Поляков, Е. Еремин. – М.: БИНОМ,Лаборатория знаний 2014. – 304 с.

[11] Поляков К.Ю. Еремин Е.А. Информатика. Углубленный уровень: учебник для 11 кл.В 2ч. Ч1 / К. Поляков, Е. Еремин. – М.: БИНОМ,Лаборатория знаний 2014. – 240 с.

[12] Поляков К.Ю. Еремин Е.А. Информатика. Углубленный уровень: учебник для 11 кл.В 2ч. Ч2 / К. Поляков, Е. Еремин. – М.: БИНОМ,Лаборатория знаний 2014. – 344 с.

[13] Романов Е.Л. Си++. От дилетанта до профессионала / Е. Романов. – 2014. – 600 с.

[14] Страуструп Б. Язык программирования С++. Специальное издание / Б. Страуструп. – М.: Москва, БИНОМ 2011. – 1136 с.

[15] Побегайло, А. П. С/С++ для студента. /Т.Павловская.- М.: БХВ-Петербург, 2006.–525 с.

Электронные ресурсы:

[16] Образовательный портал для подготовки к экзаменам Решу ЕГЭ [Электронный ресурс] – Режим доступа: `https://inf-ege.sdamgia.ru`.

[17] Основы программирования на языках Си и С++ для начинающих [Электронный ресурс] – Режим доступа: `http://cppstudio.com`.

[18] ТРЕНИРОВОЧНЫЙ КИМ - PDF [Электронный ресурс] – Режим доступа: `http://docplayer.ru`.