БАЗОВЫЕ ЭЛЕМЕНТЫ ЯЗЫКА С++

1.1. Состав языка

Алфавит языка — совокупность допустимых в языке символов. Алфавит языка С++ включает: — прописные и строчные латинские буквы и знак подчеркивания; — арабские цифры от 0 до 9, шестнадцатеричные цифры от А до F; — специальные символы "{}, | ;[]() + - / % * . \ ': ? < = >!&# у — пробельные символы: пробел, символ табуляции, символ пере хода на новую строку. Из символов алфавита формируются лексемы языка: идентификаторы, ключевые (зарезервированные) слова, знаки операций, константы, разделители (скобки, точка, запятая, пробельные символы). Границы лексем определяются другими лексемами, такими, как раз делители или знаки операций. В свою очередь, лексемы входят в состав выражений (выражение задает правило вычисления некоторого значения) и операторов (оператор задает законченное описание некоторого действия).

Идентификаторы — это имена программных объектов: констант, переменных, меток, типов, экземпляров классов, функций и полей в записи. Идентификатор может включать латинские буквы, цифры и символ подчеркивания. Прописные и строчные буквы различаются, например *myname, myName и MyName* — три различных имени. Первым символом идентификатора может быть буква или знак подчеркивания, но не цифра. Пробелы внутри имен не допускаются. Язык С++ не налагает никаких ограничений на длину имен, однако для удобства чтения и записи кода не стоит делать их слишком длинными. Ключевые слова — это зарезервированные идентификаторы, которые имеют специальное значение для компилятора, например *class, catch, int* и т.д. Ключевые слова можно использовать только по прямому назначению. С ключевыми словами и их назначением можно ознакомиться в справочной системе С++.

1.2. Структура программы

Каждая программа на языке С++ состоит из одной или несколько функций. Каждая функция содержит четыре основных элемента:

1) тип возвращаемого значения;

2) имя функции;

3) список параметров, заключенный в круглые скобки (он может быть пустым);

4) тело функции (последовательность операторов), которое поме щается в фигурные скобки.

При запуске программы вызывается функция с именем main — это главная функция программы, которая должна присутствовать в про грамме обязательно. Приведем пример простейшей функции main():

int main() {

return 0;

}

В приведенном примере:

1) тип возвращаемого значения int — целый;

2) имя функции main;

3) список параметров пуст;

4) тело функции состоит из одного оператора return 0, который воз вратит значение 0 вызывающему окружению (либо компилятору, либо операционной системе).

```
#include <iostream> // директива препроцессора
/* пример программы, подсчитывающей сумму двух целых чисел, значения которых вводятся с клавиатуры, а результат выводится на экран */ using namespace std; int main() {
int a,b; // описание переменных
cout << "Введите 2 целых числа" « endl; // оператор вывода
cin >> a >> b; ; // оператор ввода
cout << "Их сумма равна " << a + b; // оператор вывода
return 0:
}
```

Рассмотрим первую строку кода: #include — это директива препроцессора, которая указывает компилятору на необходимость

подключить перед компиляцией содержимое заголовочного файла библиотеки подпрограмм C++. Имя заголовочного файла помещено в скобки, в данном случае это iostream. В этом файле содержатся готовые программные средства для организации форматированного ввода- вывода. В общем случае в программе может быть несколько директив, каждая из которых должна начинаться с новой строки и располагаться вне тела функции.

Далее идет комментарий. Комментарии используются для кратких заметок об используемых переменных, алгоритме или для дополни тельных разъяснений сложного фрагмента кода, т.е. помогают понять смысл программы. Комментарии игнорируются компилятором, поэтому они могут использоваться для «скрытия» части кода от компилятора. В языке C++ существуют два вида комментариев: 1) комментарий, содержание которого умещается на одной строке, начинается с символов «//» и заканчивается символом перехода на новую строку; 2) если содержание комментария не умещается на одной строке, то используется парный комментарий, который заключается между символами «/* */». Внутри парного комментария могут располагаться любые символы, включая символ табуляции и символ новой строки. Парный комментарий не допускает вложения.

В четвертой строке содержится директива *using namespace std*, которая означает, что все определенные ниже имена в программе будут относиться к пространству имен std. Пространством имен называется область программы, в которой определена некоторая совокупность имен. Эти имена неизвестны за пределами данного пространства имен. В нашем случае это означает, что глобальные (т.е. определенные вне программы имена из библиотеки iostream) имена *cout, cin u endl* определены внутри пространства имен *std*, более того, в нем определены собственные имена a, b. Пространство имен *std* — это пространство имен, используемое стандартной библиотекой. Программист вправе определить собственное пространство имен.

Седьмая строка начинается с идентификатора *cout*, который представляет собой объект C++, предназначенный для работы со стандартным

потоком вывода, ориентированным на экран. Далее идет операция < < , которая называется операцией вставки или помещения в поток. Данная операция копирует содержимое, стоящее справа от символов < < < на экране появляется строка «Введите два целых числа», и курсор перемещается на новую строчку. Перемещение курсора на новую строку происходит за счет использования специализированного слова *endl*, называемого манипулятором: при его записи в поток вывода происходит перевод сообщения на новую строку. Следующая строка начинается с идентификатора *cin*. Идентификатор *cin* представляет собой объект C++, предназначенный для работы со стандартным потоком ввода, ориентированным на клавиатуру. Операция > > называется операцией извлечения из потока: она читает значения из объекта *cin* и сохраняет их в переменных, стоящих справа от символов >>. В результате выполнения данной операции с клавиатуры будут введены два значения, первое из которых сохраняется в переменной a, второе — в переменной b. Далее идет оператор вывода, в результате выполнения которого на экране появится сообщение «Их сумма равна ...» Например, если с клавиатуры были введены числа 3 и 4, то на экране появится сообщение «Их сумма равна 7»

1.3. Стандартные типы данных C++

Данные — это формализованное (понятное для компьютера) представление информации. В программах данные фигурируют в качестве значений переменных или констант. Данные, которые не изменяются в процессе выполнения программы, называются константами. Данные, объявленные в программе и изменяемые в процессе ее выполнения, называются переменными. Особенности представления данных:

1) каждое значение (переменной, константы и результата, возвращаемого функцией) имеет свой тип;

2) тип переменной или константы объявляется при их описании;

3) тип определяет:

• внутреннее представление данных в памяти компьютера,

• объем оперативной памяти, необходимой для размещения значения данного типа,

• множество значений, которые могут принимать величины этого типа,

• операции и функции, которые можно применять к величинам этого типа.

Все типы данных можно разделить на простые и составные. К простым относятся: стандартные (целые, вещественные, символьные, логический) и определенные пользователем (перечислимые типы). К составным типам относятся массивы, строки, объединения, структуры, файлы и объекты. Кроме этого существует специальный тип *void*, который не предназначен для хранения значений и применяется обычно для определения функций, которые не возвращают значения. В данном параграфе мы рассмотрим только стандартные типы данных. Целые типы данных. Существует семейство целых типов данных — *short, int, long*; и два спецификатора — *signed u unsigned*.

Таблица 1.1

Тип	Диапазон значений	Размер, байт
short	–32 768—32 767	2
unsigned short	0—65 535	2
int	–32 768—32 767 или –2 147 483 648—2 147 483 647	2, 4 или 8
unsigned int	0—65 535 или 0—4 294 967 295	2, 4 или 8
long	–2 147 483 648—2 147 483 647	4
unsigned long	0—4 294 967 295	4
long int	-2^{31}—$(2^{31} - 1)$	8

1.4. Константы

Константами называются данные, которые не изменяются в процессе выполнения программы. В C++ можно использовать именованные и неименованные константы. Неименованные константы или литералы — это обычные фиксированные значения. Например, в операторе: a = b + 2.5; //2.5- неименованная константа. Различаются целые, вещественные, символьные и строковые литералы. Компилятор относит литерал к одному из типов данных по его внешнему виду. Для записи целочисленного литерала можно использовать одну из трех форм: десятичную, восьмеричную или шестнадцатеричную. Десятичная форма: последовательность десятичных цифр, начинающаяся не с нуля, если это не число нуль. Восьмеричная: нуль, за которым следуют восьмеричные цифры (0, 1, 2, 3, 4, 5, 6, 7). Шестнадцатеричная: Ox или OX, за которыми следуют шестнадцатеричные цифры (0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F).

Символьный литерал — это один или два символа, заключенные в апострофы. Например, /', '3 ', '\п'. Большинство символьных литералов являются печатаемыми символами, т.е. они отображаются на экране в том виде, в котором записаны в апострофах. Некоторые символы являются непечатаемыми, т.е. никак не отображаются ни на экране, ни при печати. Для ввода непечатаемых символов используется управляющая последовательность. Управляющая последовательность начинается символом наклонной черты влево «\». Некоторые управляющие последовательности C++ приведены в табл. 1.3.

Таблица 1.3.

Вид	Назначение	Вид	Назначение
\a	Звуковой сигнал, оповещение (*alert*)	\t	Горизонтальная табуляция (*horizontal tab*)
\b	Возврат на 1 символ (*backspace*)	\v	Вертикальная табуляция (*vertical tab*)

Окончание табл. 1.3

Вид	Назначение	Вид	Назначение
\f	Перевод страницы (formfeed)	\\	Обратная косая черта (backslash)
\n	Перевод строки, новая строка (newline)	\'	Апостроф, одинарная кавычка (single quote)
\r	Возврат каретки (carriage return)	\"	Кавычки

Строковый литерал — это произвольное количество символов, помещенное в кавычки, например, *"Ура! Сегодня информатика!!!"*.

1.5. Переменные

Переменная — это именованная область памяти, в которой хра нятся данные определенного типа. У переменной есть имя и значение. Имя служит для обращения к области памяти, в которой хранится зна чение. Во время выполнения программы значение переменной можно изменять. Перед использованием любая переменная должна быть опи сана. Формат описания (объявления) переменных:

[<класс памяти>] <тип> <имя> [= <выражение> | (<выражение>)];

Например, опишем переменные *i, j* типа *int* и переменную *x* типа *double*:

```
int i, j;
double x;
```

Перед использованием значение любой переменной должно быть определено. Это можно сделать с помощью:

1) оператора присваивания

```
int a; // описание переменной
a = 10; // определение значения переменной
```

2) оператора ввода

```
int a; // описание переменной
cin>> a; // определение значения переменной
```

3) инициализации — определения значения переменной на этапе описания. Язык C++ поддерживает две формы инициализации пере менных: инициализация копией и прямая инициализация. Например

int i = 100; // инициализация копией
int i (100); // прямая инициализация

При одном описании можно инициализировать две и более переменных, задавая каждой собственное значение. Кроме того, в одном описании могут находиться как инициализированные, так и неинициализированные переменные. Например:

int i, day = 3, year = 2007;

1.6. Организация консольного ввода-вывода данных

Напомним, что в C++ для организации консольного ввода-вывода данных (ввод- вывод в режиме «черного» окна) необходимо подключить заголовочный файл iostream. В этом файле определены:

1) объект *cin*, который предназначен для ввода данных со стандарт ного устройства ввода — клавиатуры;

2) объект *cout*, который предназначен для вывода данных на стандартное устройство вывода — экран;

3) операция >>, которая используется для извлечения данных из входного потока;

4) операция <<, которая используется для помещения данных в выходной поток;

5) операции форматированного ввода-вывода.

Рассмотрим некоторые операции форматирования выходного потока. По умолчанию данные, помещаемые в выходной поток, форма тируются следующим образом:

1) значение типа **char** помещается в поток как единичный символ и занимает в потоке поле, размерность которого равна единице;

2) строка помещается в поток как последовательность символов и занимает в потоке поле, размерность которого равна длине строки;

3) значение целого типа помещается в поток как десятичное целое число и занимает в потоке поле, размерность которого достаточна для размещения всех цифр числа, а в случае отрицательного числа еще и для знака «минус»; положительные числа помещаются в поток без знака;

4) значение вещественного типа помещается в поток с точностью семь цифр после «десятичной запятой»; в зависимости от величины числа оно может быть выведено в экспоненциальной форме (если число очень маленькое или очень большое) или десятичной форме.

Для форматирования потоков используются манипуляторы. Мы уже рассматривали манипулятор endl, который позволяет при выводе потока на экран перенести фрагмент потока на новую строку. Теперь рассмотрим манипуляторы, которые позволяют управлять форматом вещественных типов данных и их размещением на экране.

Управление форматом вещественных типов данных. Существуют три аспекта оформления значений с плавающей запятой, которыми можно управлять: точность (количество отображаемых цифр); форма записи (десятичная или экспоненциальная); указание десятичной точки для значений с плавающей запятой, являющихся целыми числами.

Точность задает общее количество отображаемых цифр. При отображении значения с плавающей запятой округляются до текущей точности, а не усекаются. Изменить точность можно с помощью манипулятора *setprecision*. Аргументом данного манипулятора является устанавливаемая точность.

Рассмотрим следующий пример:

```
#include <iostream>
#include <iomanip>
using namespace std;
int main()
{
  double i = 12345.6789;
  cout << setprecision(3) << i << endl;
  cout << setprecision(6) << i << endl;
  cout << setprecision(9) << i << endl;
  return 0;
}
```

Результат работы программы:

```
1.23e+004
12345.7
12345.6789
```

Управление размещением данных на экране. Для размещения отображаемых данных используются манипуляторы:

1) *left* — выравнивает вывод по левому краю;

2) *right* — выравнивает вывод по правому краю;

3) *internal* — контролирует размещение отрицательного значения: выравнивает знак по левому краю, а значение по правому, заполняя пространство между ними пробелами;

4) *setprecision(int w)* — устанавливает максимальное количество цифр в дробной части для вещественных чисел в формате с фиксиро ванной точкой (манипулятор fixed) или общее количестов значащих цифр для чисел в экспоненциальном формате (манипулятор scientific);

5) *setw(int w)* — устанавливает максимальную ширину поля вывода. Рассмотрим примеры использования данных манипуляторов.

```
#include <iostream>
#include <iomanip>
using namespace std;
int main() {
cout <<"1." « setw(10) << "Ivan()v" « endl;
cout<< "2." << setw(10) << left « "Ivan()v" << endl;
cout << "3." << setw(10) « right « "Ivan()v" << endl;
```

```
return 0;
}
```

1.7. Операции

Унарные операции. Операции увеличения и уменьшения на единицу (++ и -) называются также инкрементом и декрементом соответственно. Они имеют две формы записи — префиксную, когда операция записывается перед операндом, и постфиксную — операция записывается после операнда. Префиксная операция инкремента (декремента) увеличивает (уменьшает) свой операнд и возвращает измененное значение как результат. Постфиксные версии инкремента и декремента возвращают первоначальное значение операнда, а затем изменяют его.

Рассмотрим эти операции на примере.

```
#include <iostream>
using namespace std;
int main()
{
  int x = 3, y = 4;
  cout << ++x << "\t" << --y << endl;
  cout << x++ << " \t" << y-- << endl;
  cout << x << "\t" << y << endl;
  return 0;
}
```

Операция определения размера *sizeof* предназначена для вычисле ния размера выражения или типа в байтах и имеет две формы*: sizeof , sizeof ()*. При применении операции *sizeof* к выражению возвращается размер типа, который получается в результате вычисления выражения. При применении к типу возвращается размер заданного типа. Рассмотрим данную операцию на примере.

```
#include <iostream>
using namespace std;
int main()
{
  int x = 3;
  cout << sizeof (int) << endl;
  cout << sizeof (x * 10) << endl;
  cout << sizeof (x * 0.1) << endl;
  return 0;
}
```

Логическое отрицание (!) дает в результате значение 0 (ложь), если операнд отличен от нуля (истина), и значение 1 (истина), если операнд равен нулю (ложь). Тип операнда может быть логическим, целочислен ным, вещественным или указателем. Рассмотрим данные операции на примере.

```
#include <iostream>
using namespace std;
int main() {
 int x = 3, y = 0;
bool f = false, v = true;
cout « -x << "\t" « !x « endl;
cout « -y « "\t" « !y « endl;
cout « f << "\t" « !f << endl;
cout << v<< "\t" « !v << endl;
return 0;
}
```

Бинарные операции. Арифметические операции — операции, применимые ко всем арифметическим типам (например к целым и вещественным типам), а также к любым другим типам, которые могут быть преобразованы в арифметические (например символьным типам). К бинарным арифметическим операциям относятся (в порядке убывания приоритета):

1) умножение (*), деление (/), остаток от деления (%);

2) сложение (+) и вычитание (-).

Операция *деления* применима к операндам арифметического типа. Однако, если оба операнда целочисленные, то тип результата преобразуется к целому, и в качестве ответа возвращается целая часть результата деления, в противном случае тип результата определяется правилами преобразования. Операция остаток от деления применяется только к целочисленным операндам. Знак результата зависит от реализации. Рассмотрим на примере операции деления и остаток от деления.

```
#include <iostream>
using namespace std;
int main()
{
  cout << 100/24<< "\t" << 100/24.0 << endl;
  cout << 100/21<< "\t" << 100.0/24 << endl;
  cout << 21%3<< "\t" << 21%6 << "\t" << -21%8 << endl;
  return 0;
}
```

Операции присваивания (= ,*=,/=, %= ,+=, -=). Операция простого присваивания обозначается знаком = . Формат операции простого присваивания: операнд_2 = операнд_1; В результате выполнения этой операции вычисляется значение операнда_1, и результат записывается в операнд_2. Можно связать воедино сразу несколько операторов присваивания, записывая такие цепочки:

a = b = c = 100. Присваивание такого вида выполняется справа налево: результатом выполнения c = 100 является число 100, которое затем присваивается переменной b, результатом чего опять является 100, которое присваивается переменной a.

Кроме простой операции присваивания, существуют сложные операции присваивания, например умножение с присваиванием (*=), деление с присваиванием (/=), остаток от деления с присваиванием (%=), сложение с присваиванием (+=), вычитание с присваиванием (-=) и т.д.

В сложных операциях присваивания, например, при сложении с присваиванием, к операнду_2 прибавляется операнд_1 и результат записывается в операнд_2, т.е. выражение c += a является более компактной записью выражения c = c + a. Кроме того, сложные операции присваивания позволяют сгенерировать более эффективный код за счет того, что в простой операции присваивания для хранения значения правого операнда создается временная переменная, а в сложных операциях присваивания значение правого операнда сразу записывается в левый операнд.

Тернарная операция. Условная операция (? :). Формат условной операции:

(операнд_1) ? операнд_2 : операнд_3;

Операнд_1 — это логическое или арифметическое выражение. Он оценивается с точки зрения его эквивалентности константам true (истина) и false (ложь). Если результат вычисления операнда_1 равен истине, то результатом условной операции будет значение операнда_2, иначе — операнда_3. Вычисляется всегда либо операнд_2, либо операнд_3. Их тип может различаться. Условная операция является аналогом условного оператора if, который будет рассмотрен позже. Рассмотрим тернарную операцию на примере.

```
#include <iostream>
using namespace std;
int main()
{
  int x, y, max;
  cin >> x >> y;
  (x > y) ? cout << x : cout << y << endl; // 1
  max = (x > y) ? x : y;                   // 2
  cout << max << endl;
  return 0;
}
```

Обратите внимание на то, что строки 1 и 2 решают одну и ту же задачу: находят наибольшее значение из двух целых чисел. Но в строке 2 в зависимости от условия (x > y) условная операция записывает в переменную max либо значение x, либо значение y, после чего значение переменной max можно неоднократно использовать. В строке 1 наибольшее значение просто выводится на экран, и в дальнейшем это значение использовать будет нельзя, так как оно не было сохранено ни в какой переменной.

1.8. Выражения и преобразование типов

Выражение — это синтаксическая единица языка, определяющая способ вычисления некоторого значения. Выражения состоят из опе рандов, операций и скобок. Каждый операнд является в свою очередь выражением или одним из его частных случаев — константой, переменной или функцией.

Преобразование типов в выражениях происходит неявно (без участия программистов) следующим образом: если их операнды имеют различные

типы, то операнд с более «низким» типом автоматически будет преобразован к более «высокому» типу. Тип bool в арифметических выражениях преобразуется к типу int, при этом константа true заменяется единицей, а false — нулем.

В общем случае неявные преобразования могут происходить с потерей точности или без потери точности. Рассмотрим небольшой пример.

```
#include <iostream>
using namespace std;
int main()
{
  int a = 100, b;
  float c = 4.5, d;
  d = a/c;            // 1 - без потери точности
  cout << "d = " << d << endl;
  b = a/c;            // 2 - с потерей точности
  cout << "b = " << b << endl;
  return 0;
}
```

В строке 1 переменная *a* (тип *int*) делится на переменную с (тип *float*). В соответствии с иерархией типов результат операции деления будет преобразован к типу *float*. Полученное значение записывается в переменную d (тип *float*). Таким образом, в строке 1 было выполнено одно преобразование от «низшего» типа к «высшему» и мы получили точный результат.

В строке 2 в результате операции деления также было получено значение, тип которого float. Но этот результат был записан в переменную b (тип int). Поэтому произошло еще одно преобразование — от «высшего» типа к «низшему», что повлекло потерю точности.

Рассмотренные преобразования происходили неявно (автоматически), т.е. без участия программиста. Однако программист может совершать подобные преобразования сам с помощью операций явного преобразования типов. Для демонстрации этих операций рассмотрим небольшой пример.

```
#include <iostream>
using namespace std;
int main()
{
  int a = 1500000000;
  a = (a * 10) / 10; // 1 - неверный результат
  cout << "a = " << a << endl;
  int b = 1500000000;
  b = (static_cast<double>(b) * 10) / 10; // 2 - верный результат
  cout << "b = " << b << endl;
  return 0;
}
```

1.9. Примеры простейших программ

1. Составить программу, которая для заданного значения x вычисляет значение выражения $\frac{x^2+\sin(x+1)}{25}$.

```
#include <iostream>
#include <iomanip>
#include <math.h>
using namespace std;
int main()
{
  int x;
  double y;
  cin >> x;
  y = (pow(x, 2) + sin(x + 1))/25;   // 1
  cout << "y = " << setprecision(5) << y << endl;
  return 0;
}
```

2. Написать программу, подсчитывающую площадь квадрата, периметр которого равен p.

Указания по решению задачи. Прежде чем составить программу, проведем математические рассуждения. Пусть дан квадрат со стороной a, тогда:

периметр вычисляется по формуле $p = 4a$

площадь вычисляется по формуле $s = a^2$

$$\Rightarrow \begin{matrix} a = \frac{p}{4} \\ a = \sqrt{s} \end{matrix} \Rightarrow \frac{p}{4} = \sqrt{s} \Rightarrow s = \left(\frac{p}{4}\right)^2.$$

```
#include <iostream>
#include <math.h>
using namespace std;
int main()
{
  float p, s;
  cout << "Введите периметр квадрата: ";
  cin >> p;
  s = pow(p/4, 2);
  cout << "Площадь данного квадрата = " << s;
  return 0;
}
```

3. Определить, является ли целое число четным.

Указание по решению задачи. Напомним, что число является четным, если остаток от деления данного числа на 2 равен нулю.

```
#include <iostream>
using namespace std;
int main()
{
  int x;
  cout << "Введите x";
  cin >> x;
  (x % 2 == 0)? cout << "четное\n": cout << "нечетное\n";  // 1
  return 0;
 }
```

Упражнения

I. Написать программу, которая вычисляет значение выражения.

1) $10\sin x+|x^4-x^5|$;
2) $e^{-x}-\cos x+\sin 2xy$;
3) $\sqrt{x^4+\sqrt{|x+1|}}$;
4) $\dfrac{\sin x+\cos y}{\operatorname{tg} x}+0{,}43$;
5) $\dfrac{0{,}125x+|\sin x|}{1{,}5x^2+\cos x}$;
6) $\dfrac{x+y}{x+1}-\dfrac{xy-12}{34+x}$;
7) $\dfrac{\sin x+\cos y}{\cos x-\sin y}\operatorname{tg} xy$;
8) $\dfrac{1+e^{y-1}}{1+x^2\,|\,y-\operatorname{tg} x\,|}$;
9) $|x^3-x^2|-\dfrac{7x}{x^3-15x}$;
10) $1+\dfrac{x}{3}+|x|+\dfrac{x^3+4}{2}$;
11) $\dfrac{\ln|\cos x|}{\ln(1+x^2)}$;
12) $\dfrac{1+\sin\sqrt{x+1}}{\cos(12y-4)}$;
13) $\dfrac{a^2+b^2}{1-\dfrac{a^3-b}{3}}$;
14) $\dfrac{1+\sin^2(x+y)}{2+\left|x-\dfrac{2x}{1+x^2y^2}\right|}+x$;
15) $x\cdot\ln x+\dfrac{y}{\cos x-\dfrac{x}{3}}$;
16) $\sin\sqrt{x+1}-\sin\sqrt{x-1}$;
17) $\dfrac{\cos x}{\pi-2x}+16x\cdot\cos xy$;
18) $2\operatorname{ctg}3x-\dfrac{1}{12x^2+7x-5}$;
19) $\dfrac{b+\sqrt{b^2+4ac}}{2a}-a^3+b^{-2}$;
20) $\ln\left|(y-\sqrt{|x|})\left(x-\dfrac{y}{x+\dfrac{x^2}{4}}\right)\right|$.

II. Написать программу, которая подсчитывает:

1) периметр квадрата, площадь которого равна a;
2) площадь равностороннего треугольника, периметр которого равен p;
3) расстояние между точками с координатами a, b и c, d;
4) среднее арифметическое кубов двух данных чисел;
5) среднее геометрическое модулей двух данных чисел;
6) гипотенузу прямоугольного треугольника по двум данным катетам a, b;
7) площадь прямоугольного треугольника по двум катетам a, b;
8) периметр прямоугольного треугольника по двум катетам a, b;
9) ребро куба, площадь полной поверхности которого равна s;
10) ребро куба, объем которого равен v;
11) периметр треугольника, заданного координатами вершин $x1, y1, x2, y2, x3, y3$;
12) площадь треугольника, заданного координатами вершин $x1, y1, x2, y2, x3, y3$;
13) радиус окружности, длина которой равна l;
14) радиус окружности, площадь круга которой равна s;
15) площадь равнобедренной трапеции с основаниями a и b и углом α при большем основании;

16) площадь кольца с внутренним радиусом $r1$ и внешним $r2$;

17) радиус окружности, вписанной в равносторонний треугольник со стороной a;

18) радиус окружности, описанной около равностороннего треугольника со стороной a;

19) сумму членов арифметической прогрессии, если известны ее первый член, разность и число членов прогрессии;

20) сумму членов геометрической прогрессии, если известны ее первый член, знаменатель и число членов прогрессии.

III. Написать программу, которая определяет:

1) максимальное значение для двух различных вещественных чисел;
2) является ли заданное целое число четным;
3) является ли заданное целое число нечетным;

4) если целое число M делится на целое число N, то на экран выводится частное от деления, в противном случае выводится сообщение «M на N нацело не делится»;

5) оканчивается ли данное целое число цифрой 7;

6) имеет ли уравнение $ax^2 + bx + c = 0$ решение, где a, b, c — данные вещественные числа;

7) какая из цифр двухзначного числа больше: первая или вторая;
8) одинаковы ли цифры данного двухзначного числа;
9) является ли сумма цифр двухзначного числа четной;
10) является ли сумма цифр двухзначного числа нечетной;
11) кратна ли трем сумма цифр двухзначного числа;
12) кратна ли числу A сумма цифр двухзначного числа;
13) какая из цифр трехзначного числа больше: первая или последняя;
14) какая из цифр трехзначного числа больше: первая или вторая;
15) какая из цифр трехзначного числа больше: вторая или последняя;
16) все ли цифры трехзначного числа одинаковые;
17) существует ли треугольник с длинами сторон a, b, c;
18) является ли треугольник с длинами сторон a, b, c прямоугольным;
19) является ли треугольник с длинами сторон a, b, c равнобедренным;
20) является ли треугольник с длинами сторон a, b, c равносторонним.