КАЗАНСКИЙ (ПРИВОЛЖСКИЙ) ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ
ИНСТИТУТ МАТЕМАТИКИ И МЕХАНИКИ ИМ. Н.И. ЛОБАЧЕВСКОГО
Кафедра теории функций и приближений

С.В. МАКЛЕЦОВ

ОСНОВЫ КОМПЬЮТЕРНЫХ НАУК
Часть 1

Учебное пособие

Казань — 2015

УДК 004.43

ББК 32.973.26-018

Печатается по решению методической комиссии

Института математики и механики им. Н.И. Лобачевского

Протокол № 9 от «18» июня 2015 г.

Заседания кафедры теории функций и приближений ИМиМ КФУ

Протокол № 10 от 4 июня 2015 г.

Рецензенты:

кандидат физико-математических наук,
доцент кафедры теории функций и приближений КФУ **Е.К. Липачёв**;
кандидат физико-математических наук,
доцент кафедры прикладной математики КФУ **Д.Н. Тумаков**

Маклецов С.В.
Основы компьютерных наук. Часть 1 / С.В. Маклецов. — Казань: Казан. ун-т, 2015. — 116 с.

В настоящем пособии приведена программа первой части курса «Основы компьютерных наук» для бакалавров 1 курса Института математики и механики им. Н.И. Лобачевского Казанского (Приволжского) федерального университета, обучающихся по направлению 02.03.01 — «Основы компьютерных наук». Пособие также включает теоретические сведения по изучаемым темам, примеры разобранных практических заданий, упражнения для выполнения самостоятельной работы, а также список рекомендуемой литературы.

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Целями освоения дисциплины (модуля) «Основы компьютерных наук» являются подготовка в области применения современной вычислительной техники для решения практических задач обработки данных, математического моделирования, информатики; получение высшего профессионального (на уровне бакалавра) образования, позволяющего выпускнику успешно работать в избранной сфере деятельности с применением современных компьютерных технологий. Для изучения и освоения дисциплины требуются первоначальные знания из курсов математического анализа, алгебры, аналитической геометрии. Знания и умения, приобретенные студентами в результате изучения дисциплины, будут использоваться при изучении курсов численных методов, вычислительного практикума, при выполнении курсовых и выпускных квалификационных работ, связанных с математическим моделированием и компьютерной обработкой данных.

Уважаемые студенты! В рамках курса «Основы компьютерных наук» вам предстоит познакомиться с работой в пакете Wolfram Mathematica, предназначенном для осуществления математических вычислений, а также с основами классических языков программирования высокого уровня «Си» и «Си++» (C/C++).

Для успешного освоения материала и приобретения практики программирования вам потребуется установить пакет Wolfram Mathematica версии 8.0 или выше, а также среду разработки MS Visual Studio версии не ниже 2012.

Настоящее пособие содержит учебную программу первой части курса «Основы компьютерных наук», а также теоретический материал, примеры практических заданий и упражнения для самостоятельного выполнения по каждой из тем учебной программы.

В процессе изучения курса вы научитесь:

- принципам использования математического пакета Wolfram Mathematica;

- созданию консольных приложений на языке программирования высокого уровня, содержащих основные алгоритмические конструкции;
- созданию приложений для решения вычислительных задач;
- осуществлению анализа существующих алгоритмов, записанных на языке программирования высокого уровня.

РАБОЧАЯ ПРОГРАММА ПЕРВОЙ ЧАСТИ КУРСА «ОСНОВЫ КОМПЬЮТЕРНЫХ НАУК»

1 семестр

Тема 1. Математические программные пакеты, их использование при решении прикладных задач.

В рамках темы изучаются:

- основы работы в пакете Wolfram Mathematica: вычислительные операции, работа с выражениями, построение графиков функций, создание наборов данных;
- элементы программирования в пакете Wolfram Mathematica.

На изучение темы отводится **2** часа лекционных занятий и **8** часов лабораторных работ.

Тема 2. Языки программирования высокого уровня. Основные операторы и типы данных. Консольный ввод/вывод.

В рамках темы изучаются:

- основные сведения о языках программирования высокого уровня;
- синтаксис языка Си;
- структура программы;
- основные типы данных (int, float, double, char, long и т.д.);
- организация хранения данных в памяти компьютера;
- организация ввода/вывода данных;
- основные видов алгоритмов (линейные, разветвляющиеся, циклические).

На изучение темы отводится **4** часа лекционных занятий и **6** часов лабораторных работ.

Тема 3. Основы процедурного программирования. Функции. Шаблоны функций. Разделение кода по файлам.

В рамках темы изучаются:

- разработка собственных дополнительных функций;
- рекурсивные функции;
- заголовочные файлы и распределение функций по файлам;
- перегрузка функций;
- шаблоны функций.

На изучение темы отводится **4** часа лекционных занятий и **6** часов лабораторных работ.

Тема 4. Работа с памятью. Организация хранения наборов данных в программах. Способы передачи параметров в функции и возвращения результатов.

В рамках темы изучаются:

- указатели и ссылки, адресная арифметика;
- передача параметров в функции по указателям и ссылкам;
- создание статических массивов;
- создание динамических массивов, их связь с указателями;
- создание функций с переменным числом параметров;
- указатели на функции;
- алгоритмы сортировки и поиска данных в массивах;
- создание функций для работы с массивами.

На изучение темы отводится **4** часа лекционных занятий и **10** часов лабораторных работ.

Тема 5. Работа с нуль-терминальными строками.

В рамках темы изучаются:

- функции для работы со строками;
- преобразование типов;
- работа со строкой как с массивом символов;
- преобразование строк и организации поиска в строках.

На изучение темы отводится **4** часа лекционных занятий и **6** часов лабораторных работ.

1 СЕМЕСТР

ТЕМА 1. МАТЕМАТИЧЕСКИЕ ПРОГРАММНЫЕ ПАКЕТЫ, ИХ ИСПОЛЬЗОВАНИЕ ПРИ РЕШЕНИИ ПРИКЛАДНЫХ ЗАДАЧ.

А. ТЕОРЕТИЧЕСКАЯ СПРАВКА

Вычислительная система Wolfram Mathematica известна как мощнейшая математическая платформа. Программный пакет применяется повсеместно при расчетах от ежедневных студенческих задачек, до ультрасовременных научных исследований.

Работа с пакетом базируется на диалоговом интерфейсе: пользователь набирает команду в так называемую ячейку, а приложение выводит результат. Ячейки обозначаются в правой части окна документа синими скобками (Желтый цвет скобки сигнализирует о производимых в данный момент вычислениях). Введенные команды помечаются как входные данные (In) и снабжаются номером, а результаты — как выходные данные (Out), также снабженные порядковым номером (см. рис. 1). Эти номера могут быть использованы для указания ссылок на ранее полученные результаты.

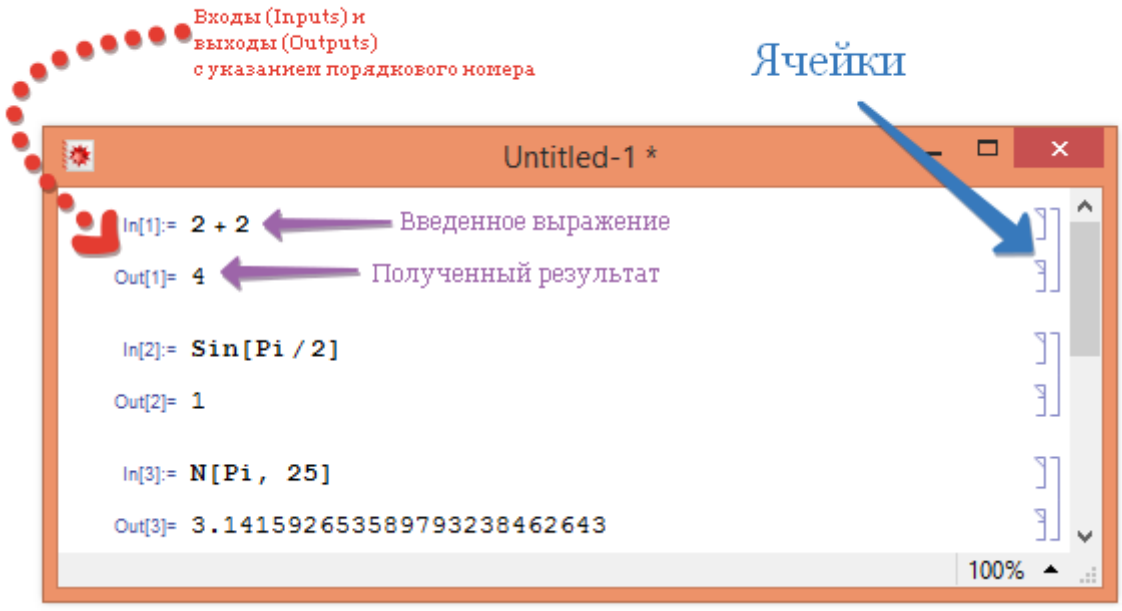


Рис. 1

Обратите внимание! Нумерация не сохраняется с файлом и ведется во время сеанса работы с пакетом. При переоткрытии приложения, нумерация всех действий начинается заново.

Чтобы после ввода команды получить результат, необходимо нажать на клавиатуре комбинацию клавиш Ctrl+Enter, либо одну клавишу Enter, расположенную в цифровом блоке клавиатуры.

При работе с пакетом Mathematica важно запомнить основные правила записи команд.

1. Для выполнения арифметических действий используется стандартный вариант их записи; операции выполняются с учетом приоритета. Изменение последовательности выполнения операций производится посредством круглых скобок.

   Обратите внимание! Символ «*» можно заменять пробелом для выполнения операции умножения (например, запись «x y» будет трактоваться как «x*y»). Также при умножении константы на переменную можно не ставить и пробела (например, выражение «2x» будет воспринято как «2*x»).

2. Название функций (например, Sin, Cos, Log и т.д.) и стандартных констант (например, Pi, E) пишутся с заглавной буквы. Если имя составное, то каждая часть названия пишется с заглавной буквы (например, ArcSin, ParametricPlot).
3. Аргументы функции и только они записываются в квадратных скобках (например, Sin[Pi/2]). Если функция принимает несколько аргументов, они разделяются запятой (например, Log[10, 100]).
4. Данные, входящие в некоторый набор (например, списки, элементы матрицы или указание переменной и пределов ее изменения), перечисляются через запятую внутри фигурных скобок.

Ниже приводятся таблицы со списком некоторых функций и операторов пакета Mathematica.

Таблица 1.

Список основных операторов пакета Mathematica

Оператор	Описание
=	Присвоение левому операнду значения, находящемуся справа от знака присвоения. Значение вычисляется немедленно.
:=	Отложенное присваивание. В отличие от предыдущего оператора, выражение в правой части вычисляется не сазу, а каждый раз при обращении к переменной, находящей в левой части.
+	Сложение двух операндов.
+=	Сложение операндов с присвоением результата левому операнду.
++	Увеличение операнда на 1.
-	Вычитание.
-=	Вычитание с присвоением.
--	Уменьшение операнда на 1.
*	Умножение.
*=	Умножение с присвоением.
/	Деление.
/=	Деление с присвоением.
^	Возведение в степень .
!	Вычисление факториала числа.
!!	Вычисление полуфакториала (произведение всех четных (для четного операнда) или нечетных чисел меньших либо равных данному).
.	Вычисление произведения двух матриц по правилам матричного умножения.
=.	Очистка значения переменной, находящейся слева от оператора.
%	Ссылка на предыдущий результат.
%%...%	Ссылка на результат, полученный два или более (в зависимости от количества знаков «%») шагов назад.
%a	Ссылка на результат, полученный в ячейке Out с номером a.
/.	Подстановка значения в выражение. Например, x^2 /. x->5 (Ответ: 25).
( )	Круглые скобки для указания приоритета операций.
[ ]	Квадратные скобки являются признаком функции и содержат ее аргументы.
[[i]]	Двойные квадратные скобки используются для получения i-го значения из списка данных.
{ }	Фигурные скобки используются для создания списков, перечислений, матриц.
f //N	Запись выражения f в числовом виде.

Таблица 2.

Список некоторых элементарных функций пакета Mathematica

Функция	Описание
Abs[x]	Вычисление абсолютного значения действительного числа и модуля комплексного числа.
Exp[x]	Вычисление экспоненциальной функции.
Log[x]	Вычисление натурального логарифма числа x.
Log[a, x]	Вычисление логарифма числа x по основанию a.
Sqrt[x]	Нахождение квадратного корня числа x.

Функция	Описание
Sign[x]	Определяет знак числа (возвращает -1 для отрицательного x, 0 — для x = 0 и 1 для положительного x). Для комплексного x возвращает отношение x/Abs[x].
Sin[x], Cos[x], Tan[x]. Cot[x]	Вычисляют значения основных тригонометрических функций (синуса, косинуса, тангенса, котангенса).
ArcSin[x]. ArcCos[x], ArcTan[x], ArcCot[x]	Вычисляют значения обратных тригонометрических функций (арксинуса, арккосинуса, арктангенса, арккотангенса).
Sinh[h], Cosh[h], Tanh[h], Coth[h]	Вычисляет значение гиперболических функций: синуса, косинуса, тангенса и котангенса.

Таблица 3.

Список арифметических функций пакета Mathematica

Функция	Описание
N[x]	Представление результата в виде десятичного числа.
N[x, e]	Получение числового представления результата x с заданным количеством знаков e.
Plus[x, y, …]	Выполняет суммирование аргументов.
Minus[x]	Производит смену знака числа.
Times[x]	Вычисляет произведение аргументов.
Divide[x, y]	Вычисляет частное от деления x на y.
Mod[x, y]	Возвращает остаток от деления x на y.
Divisors[x]	Возвращает список целочисленных делителей числа x.
GCD[x, y, …]	Находит наибольший общий делитель аргументов.
LCM[x, y, …]	Определяет наименьшее общее кратное аргументов.
Round[x]	Округление x до ближайшего целого.
Floor[x]	Возвращает наибольшее целое число, не превышающее x.
Ceiling[x]	Возвращает наибольшее число, большее или равно x.
Quotient[x, y]	Возвращает округленное целое число n/m, не превышающее значения n/m.
Factorial[n]	Вычисляет значение факториала числа n.
Factorial2[n]	Вычисляет значение двойного факториала числа n.
Prime[n]	Возвращает n-е простое число.
Re[z]	Возвращает вещественную часть комплексного числа z.
Im[z]	Возвращает мнимую часть комплексного числа z.
Arg[z]	Возвращает аргумент комплексного числа z.
Conjugate[z]	Возвращает комплексное число, сопряженное с z.

Таблица 4.

Список функций пакета Mathematica для работы с выражениями

Функция	**Описание**
Simplify[f]	Упрощает выражение f.
FullSimplify[f]	Упрощает выражение f, имеющее в своем составе специальные функции.
Expand[f]	Раскрывает скобки и расширяет выражение f.
ExpandAll[f]	Раскрывает все произведения и целые степени во всех частях выражения f.
PowerExpand[f]	Раскрывает скобки и возводит в целую положительную степень вложенные выражения функции f.
TrigExpand[f]	Раскрывает тригонометрическое выражения f.
Collect[f, x]	Представляет выражение f по степеням x.
Factor[f]	Раскладывает математическое выражение f на множители.
TrigFactor[f]	Раскладывает на множители тригонометрическое выражение f.
FactorTerms[f]	Выносит за скобки общий множитель.
TrigToExp[f]	Представляет тригонометрическое выражение в экспоненциальной форме.
ExpToTrig[f]	Представляет экспоненциальное выражение в тригонометрической форме.
Denominator[f]	Возвращает знаменатель выражения.
Numerator[f]	Возвращает числитель выражения.
Together[f]	Приводит выражение к общему знаменателю.

Таблица 5.

Список специальных функций пакета Mathematica

Функция	**Описание**
Sum[f_i, {i, i_{min}, i_{max}}]	Вычисляет сумму вида $\sum_{i=i_{min}}^{i_{max}} f_i$ с шагом между значениями индекса i равным 1.
Sum[f_i, {i, i_{max}}]	Вычисляет сумму, как в первой строке таблицы, для $i_{min} = 1$.
Sum[f_i, {i, i_{min}, i_{max}, Δ_i}]	Вычисляет сумму, как в первой строке таблицы, с шагом Δi.
Sum[$f_{i,j\ldots}$, {i, i_{min}, i_{max}}, {j, j_{min}, j_{max}},…]	Вычисляет сумму, общий член которой зависит от двух и более индексов.
NSum[…]	Вычисление сумм в численном виде (параметры функции могут быть такими же, как в первых четырех строках таблицы).
Product[…]	Вычисление произведений в аналитическом виде. Параметры функции могут быть такими же, как в первых четырех строках таблицы.
NProduct[…]	Вычисление произведений в численном виде. Параметры функции могут быть такими же, как в первых четырех строках таблицы.
Limit [f[x], x -> x_0, Direction-> d]	Вычисляет предел функции, зависящей от x при $x \to x_0$. Необязательный параметр Direction задает направление приближения к пределу. (d=+1 — слева, d=-1 — справа).

Функция	Описание
Series[f[x], {x, x_0, n}]	Разложение функции f[x] в степенной ряд в окрестности точки x_0, где n — степень многочлена.
D[f, x]	Вычисляет частную производную функции f по переменной x.
D[f, {x, n}]	Вычисляет n-ю частную производную функции f по переменной x.
D[f, x_1, x_2, …]	Возвращает производную функции f по параметрам x_1, x_2 и т.д.
Dt[f, x]	Возвращает обобщенную производную функции f по переменной x.
Dt[f]	Возвращает дифференциал функции f.
Integrate[f,x]	Вычисляет неопределенный интеграл от функции f по dx.
Integrate[f, {x, x_{min}, x_{max}}]	Вычисляет определенный интеграл от функции f в пределах от x_{min} до x_{max}.

Таблица 6.

Список функций пакета Mathematica для работы со списками и матрицами

Функция	Описание
Table[f, {n, n_{min}, n_{max}, dn}]	Создает список функций f при изменении переменной n от n_{min} до n_{max} с шагом dn.
Table[f, {n, n_{min}, n_{max}}]	Создает список функций f при изменении переменной n от n_{min} до n_{max} с шагом 1.
Table[f, {n, n_{max}}]	Создает список функций f при изменении переменной n от 1 до n_{max} с шагом dn.
Table[f, {n_{max}}]	Создает n_{max} экземпляров выражения f.
Table[f, {n, n_{min}, n_{max}}, {m, m_{min}, m_{max} }]	Создает матрицу, содержащую значения функции f при изменении переменной n от n_{min} до n_{max} и m от m_{min} до m_{max}.
Range[n_{min}, n_{max}, dn]	Генерирует вектор из числовых элементов от n_{min} до n_{max} с шагом dn.
List[a, b, c, …]	Создает вектор {a, b, c, …}.
MatrixForm[T]	Представляет список T в матричном виде.
TableForm[T]	Представляет список T в табличном виде (в отличие от матричного вида, табличный вид не содержит обрамляющих скобок).
Part[T, n]	Возвращает n-ый элемент вектора T.
VectorQ[V]	Проверяет, является ли V вектором, и выдает True, если да и False в противном случае.
MatrixQ[M]	Проверяет, является ли M матрицей, и выдает True, если да и False в противном случае.
Length[T]	Возвращает число элементов матрицы или вектора T.
MemberQ[V, n]	Проверяет, имеется ли в векторе V элемент n.
Dimensions[T]	Возвращает число элементов вектора T или число строк и столбцов матрицы T.
Position[V, n]	Возвращает номер позиции элемента n в векторе V.
Count[V, n]	Возвращает количество элементов n в векторе V.

Функция	Описание
TensorRank[T]	Выдает ранг вектора T или ранг матрицы T, если T является тензором.
Drop[V,n]	Удаляет первые n элементов вектора V.
Drop[V,-n]	Удаляет последние n элементов вектора V.
Drop[V,{n}]	Удаляет n-ый элемент вектора V.
Drop[V,{m,n}]	Удаляет элементы с m-го по n-ый из вектора V.
Last[T]	Возвращает последний элемент вектора или матрицы.
Rest[V]	Возвращает вектор V с отброшенным первым элементом.
Take[V, n]	Возвращает вектор V с первыми n элементами.
Take[V, -n]	Возвращает вектор V с последними n элементами.
Take[V, {m,n}]	Возвращает вектор V с номерами элементов от m до n.
Append[V, n]	Добавляет элемент n в конец вектора.
Prepend[V, n]	Добавляет элемент n в начало вектора.
Insert[V, a, n]	Добавляет элемент a в позицию n с отсчетом с начала вектора V.
Insert[V, a, -n]	Вставляет элемент a в позицию n с отсчетом с конца вектора V.
Sort[T]	Сортирует элементы вектора или матрицы T в естественном порядке
Reverse[T]	Возвращает вектор или матрицу с элементами, записанными в обратном порядке относительно вектора или матрицы T.
Transpose[M]	Транспонирует матрицу M.
Det[M]	Вычисляет определитель матрицы M.
IdentityMatrix[m]	Возвращает единичную матрицу размерности m.
Inverse[M]	Возвращает матрицу, обратную к M.
Tr[M]	Возвращает след матрицы M (сумму диагональных элементов).
LinearSolve[M,b]	Возвращает вектор-решение системы линейных алгебраических выражений M*x = b.

Таблица 7.

Список функций пакета Mathematica

для решения уравнений, систем и неравенств

Функция	Описание
Solve[f, x]	Решает уравнение[1] f относительно неизвестной x
Solve[{f_1, f_2, …}, {x_1, x_2, …}]	Решает систему уравнений f_1, f_2,… относительно неизвестных x_1, x_2, …
Roots[f, x]	Определяет корни полинома f относительно переменной x.
FindRoot[f, {x, x_0}]	Поиск корня уравнения f относительно неизвестной x с начальным приближением x_0.

[1] При задании уравнений знак равенства удваивается: f[x] == 0.

Функция	Описание
Reduce[f, {x, …}, dom]	Решает уравнения, системы или неравенства относительно заданных неизвестных. Домен dom определяет домен решений: Reals (вещественные числа), Integers (целые числа), Complexes (комплексные числа). Уравнения и неравенства, образующие систему записываются через знак “&&” либо “||”.
InterpolateRoot[f, {x, a, b}]	Ищет корни уравнения f(x) = 0 на основе интерполяции на отрезке [a,b]. Для работы функции необходимо предварительное подключение пакета расширения с помощью команды: <<FunctionApproximations`;

Таблица 8.

Список функций пакета Mathematica

для построения графиков

Функция	Описание
Plot[f, {x, a, b}]	Строит график функции f, зависящей от переменной x, на отрезке [a, b].
Plot[{f, g, …}, {x, a, b}]	Строит график нескольких функций f, g, … на отрезке [a, b] (на одном чертеже).
Plot[f, {x, a, b}, Options] Возможные опции:	Необязательный параметр Options функции Plot позволяет задавать дополнительные настройки для графика функции.
PlotRange->{ymin, ymax}	Задает интервал построения графика по вертикальной оси.
AxesLabel->{"x","y"}	Определяет подписи координатных осей.
PlotLabel->"F"	Задает название графика.
Axes->None	Убирает координатные оси.
PlotStyle->RGBColor[.5,.1, .5]	Задает цвет линии графика.
PlotStyle->Thikness[k]	Задает толщину линии графика
AspectRatio->Automatic	Делает график пропорциональным по осям.
ListPlot[{{x_1, y_1}, {x_2, y_2}, …}]	Строит график в виде точек на плоскости.
Show[Gr1, Gr2, …]	Совмещает два и более графиков, на одном чертеже.
PolarPlot[f, {t, t_{min}, t_{max}}]	Строит график функции в полярной системе координат.
BarChart[V]	Строит столбиковую диаграмму (гистограмму) по данные вектора V.
PieChart[D]	Строит круговую диаграмму. D — данные в виде списка, характеризующие долю круговой диаграммы.
Plot3D[f, {x, x_{min}, x_{max}}, {y, y_{min},y_{max}}]	Строит поверхность в трехмерном пространстве, заданную функцией двух переменных.
ParametricPlot3D[f_1, f_2, f_3, {u, u_{min}, u_{max}}, {v, v_{min}, v_{max}}]	Строит трехмерный график функции, заданной параметрически.
ContourPlot[f(x,y)==a, {x, x_{min}, x_{max}}, {y, y_{min}, y_{max}}]	Строит график функции, заданной неявно.

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
ФГАОУВПО КАЗАНСКИЙ (ПРИВОЛЖСКИЙ) ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ
ИНСТИТУТ МАТЕМАТИКИ И МЕХАНИКИ ИМ. Н.И. ЛОБАЧЕВСКОГО
КАФЕДРА ТЕОРИИ ФУНКЦИЙ И ПРИБЛИЖЕНИЙ

Лабораторная работа
“Исследование функции $f(x)=x^2+\frac{1}{x}$”

Работу выполнил
студент 05-509 группы
Иванов И.И.

Казань - 2015

```
F[x_] = x^2 + (1 / x);
```

1. Область определения функции.

Поскольку в функции присутствует дробь, необходимо из ее области определения исключить точки, где знаменатель обращается в ноль. Найдем их, предварительно приведя выражение к общему знаменателю:

```
Denominator[Together[F[x]]]
```

```
x
```

```
Solve[% == 0, x]
```

```
{{x → 0}}
```

Т.о. исследуемая функция определена на интервале $(-\infty; 0) \cup (0; +\infty)$

2. Исследование функции на четность/нечетность/периодичность.

Проверка на четность :

```
F[x] == F[-x]
```

$$\frac{1}{x} + x^2 == -\frac{1}{x} + x^2$$

Проверка на нечетность:

```
F[x] == -F[-x]
```

$$\frac{1}{x} + x^2 == \frac{1}{x} - x^2$$

Поскольку в обоих случаях не получен ответ “True”, функция не прошла проверку ни на четность, ни на нечетность, а значит является функцией общего вида.

Проверка на периодичность :

```
Reduce[F[x] = F[x + T], T]
```

$$(T == 0 \&\& x \neq 0) \;||\; \left(x \neq 0 \&\& \left(T == \frac{-3x^2 - \sqrt{x}\sqrt{4+x^3}}{2x} \;||\; T == \frac{-3x^2 + \sqrt{x}\sqrt{4+x^3}}{2x}\right)\right)$$

Исходя из полученного ответа следует, что не существует такого периода T≠T[x], отличного от 0 для любого допустимого значения x, для которого бы выполнялось условие F[x]=F[x+T], а значит функция F[x] не является периодической.

---

3. Нахождение точек пересечения графика исследуемой функции с осями координат.

Пересечение с осью OX (точка(и) x, где значение функции =0):

```
NSolve[F[x] = 0, x]
```

```
{{x → -1.}, {x → 0.5 + 0.866025 i}, {x → 0.5 - 0.866025 i}}
```

График функции пересекает ось OX в одной точке с действительной координатой x=-1.

Для пересечения с осью OY необходимо вычислить значение функции при x=0, однако поскольку в п.1 было показано, что точка x=0 не входит в область определения функции, можно заключить, ее график не пересекает ось OY.

---

4. Нахождение промежутков знакопостоянства функции.

Возможные точки смена знака функции - это точки пересечения с осью OX, и точки разрыва.

Точка пересечения с осью OX: x = -1 (см. пункт 3); точка разрыва функции: x = 0 (см. пункт 1).
Проверим знаки функции на промежутках (-∞; -1), (-1, 0), (0, ∞). Для этого вычислим ее значения в произвольных точках из указанных интервалов. Знак полученного значения будет одинаковым для любой точки из интервала:

```
F[-3]
```

$$\frac{26}{3}$$

```
F[-0.5]
```

```
-1.75
```

```
F[1]
```

```
2
```

Таким образом, функция положительна на интервале (-∞; -1) ⋃ (0; +∞) и отрицательна на интервале (-1; 0)

---

5. Нахождение производной, области ее определения, критических точек.

Поиск производной функции:

```
G[x_] = D[F[x], x]
```

$$-\frac{1}{x^2}+2\,x$$

Область определения производной :

```
Denominator[Together[G[x]]]
```

$$x^2$$

```
Solve[% == 0, x]
```

$$\{\{x\to 0\},\{x\to 0\}\}$$

Точки, принадлежащие области определения функции F[x], в которых ее производная не существует или равна 0 являются критическими для функции F[x].
Областью определения производной G[x] является интервал (-∞;0)⋃(0;+∞)
Однако точка x=0 не является критической для функции F[x], поскольку не принадлежит области ее определения.

Найдем другие критические точки для функции F[x] - это точки, в которых производная G[x]=0.

```
Solve[G[x] == 0, x]
```

$$\left\{\left\{x\to -\left(-\frac{1}{2}\right)^{1/3}\right\},\left\{x\to \frac{1}{2^{1/3}}\right\},\left\{x\to \frac{(-1)^{2/3}}{2^{1/3}}\right\}\right\}$$

```
% // N
```

```
{{x → -0.39685 - 0.687365 i}, {x → 0.793701}, {x → -0.39685 + 0.687365 i}}
```

Производная G[x]=0 при одном действительном значении $x = 2^{-1/3} \approx 0.793701$. Эта точка принадлежит области определения функции F[x], а значит является критической для функции F[x].

---

6. Нахождение промежутков возрастания, убывания, точек экстремума и экстремумов.

Критические точки функции разбивают область ее определения на промежутки. Промежутком возрастания является промежуток области определения функции, где ее производная положительна. Промежутком убывания - где производная отрицательна. При этом, если при переходе через критическую точку производная меняет знак, то эта критическая точка является точкой экстремума. Если знак меняется с “+“ на “-“ - это точка максимума, если с “-“ на “+“ - точка минимума.

Определим знак производной на трех интервалах: $(-\infty;0)$, $(0;2^{-1/3})$, $(2^{-1/3};\infty)$. Здесь точка 0 - точка разрыва области определения функции F[x], а $2^{-1/3}$ - критическая точка.

```
G[-1]
```

```
-3
```

```
G[0.5]
```

```
-3.
```

```
G[1]
```

```
1
```

Таким образом, на интервалах $(-\infty;0)$ и $(0;2^{-1/3})$ функция F[x] убывает, а на интервале $(2^{-1/3};\infty)$ исследуемая функция возрастает.

При этом, точка $x=2^{-1/3}$ является точкой экстремума функции. Поскольку в ней производная G[x] меняет знак с “-“ на “+“ - указанная точка является точкой ***минимума***.

Определим значение функции F[x] в этой точке:

```
x = 2^-1/3
```

$$\frac{1}{2^{1/3}}$$

```
F[x]
```

$$\frac{1}{2^{2/3}} + 2^{1/3}$$

```
% // N
```

```
1.88988
```

T.o. $F\left[2^{-1/3}\right] = \frac{1}{2^{2/3}} + 2^{1/3} \approx 1.88988$ – экстремум (***минимум***) исследуемой функции.

7. Нахождение промежутков выпуклости и точек перегиба.

Для нахождения указанных промежутков необходимо найти вторую производную функции :

```
x = .
```

```
H[x_] = D[F[x], {x, 2}]
```

$$2 + \frac{2}{x^3}$$

Точки, в которых вторая производная равна 0 или не существует, разбивают область ее определения на промежутки. Функция F[x] выпукла вниз на промежутке, если вторая ее производная на нем положительна, и выпукла вверх на промежутке, если вторая производная отрицательна. Если при переходе через точку, принадлежащую области определения функции, в которой вторая производная (H[x]) равна 0 или не существует, H[x] меняет свой знак, то данная точка является точкой перегиба.

Найдем область определения второй производной:

```
Denominator[Together[H[x]]]
```

$$x^3$$

```
Solve[% == 0, x]
```

```
{{x → 0}, {x → 0}, {x → 0}}
```

Т.о. H[x] имеет область определения ℝ\{0}

```
Solve[H[x] == 0, x]
```

$$\left\{\{x \to -1\}, \left\{x \to (-1)^{1/3}\right\}, \left\{x \to -(-1)^{2/3}\right\}\right\}$$

```
% // N
```

```
{{x → -1.}, {x → 0.5 + 0.866025 i}, {x → 0.5 - 0.866025 i}}
```

Вторая производная равна 0 в одной действительной точке x=-1.

Определим знак второй производной на трех интервалах : $(-\infty; -1)$, $(-1; 0)$, $(0; +\infty)$. Здесь точка $x = 0$ – точка разрыва области определения функции F[x], а $x = -1$ – критическая точка.

```
H[-2]
```

$$\frac{7}{4}$$

```
H[-0.5]
```

```
-14.
```

```
H[1]
```

```
4
```

Таким образом, на интервалах (-∞;-1) и (0;+∞) функция F[x] выпукла вниз, а на интервале (-1;0) - выпукла вверх. В точке x=-1 вторая производная меняет свой знак с плюса на минус, а значит эта точка является точкой перегиба.

8. Исследование поведения функции на бесконечности и в окрестностях точек разрыва.

Рассмотри точку разрыва функции F[x] - x=0 (см. пункт 1). В случае, если хотя бы один из пределов $\lim_{x\to 0+} f(x)$, $\lim_{x\to 0-} f(x)$ окажется равным ±∞, это будет означать, что в данной точке имеется вертикальная асимптота.

```
Limit[F[x], x → 0, Direction → 1]
```

```
-∞
```

```
Limit[F[x], x → 0, Direction → -1]
```

```
∞
```

Т.о. в прямая x = 0 является вертикальной асимптотой к графику функции F[x].

Для опредления поведения функции на бесконечности, выясним, существуют ли наклонные асимптоты. С этой целью найдем следующие пределы:

```
k1 = Limit[F[x] / x, x → +∞]
```

```
∞
```

```
b1 = Limit[F[x] - k1 * x, x → +∞]
```

```
∞
```

```
k2 = Limit[F[x] / x, x → -∞]
```

```
-∞
```

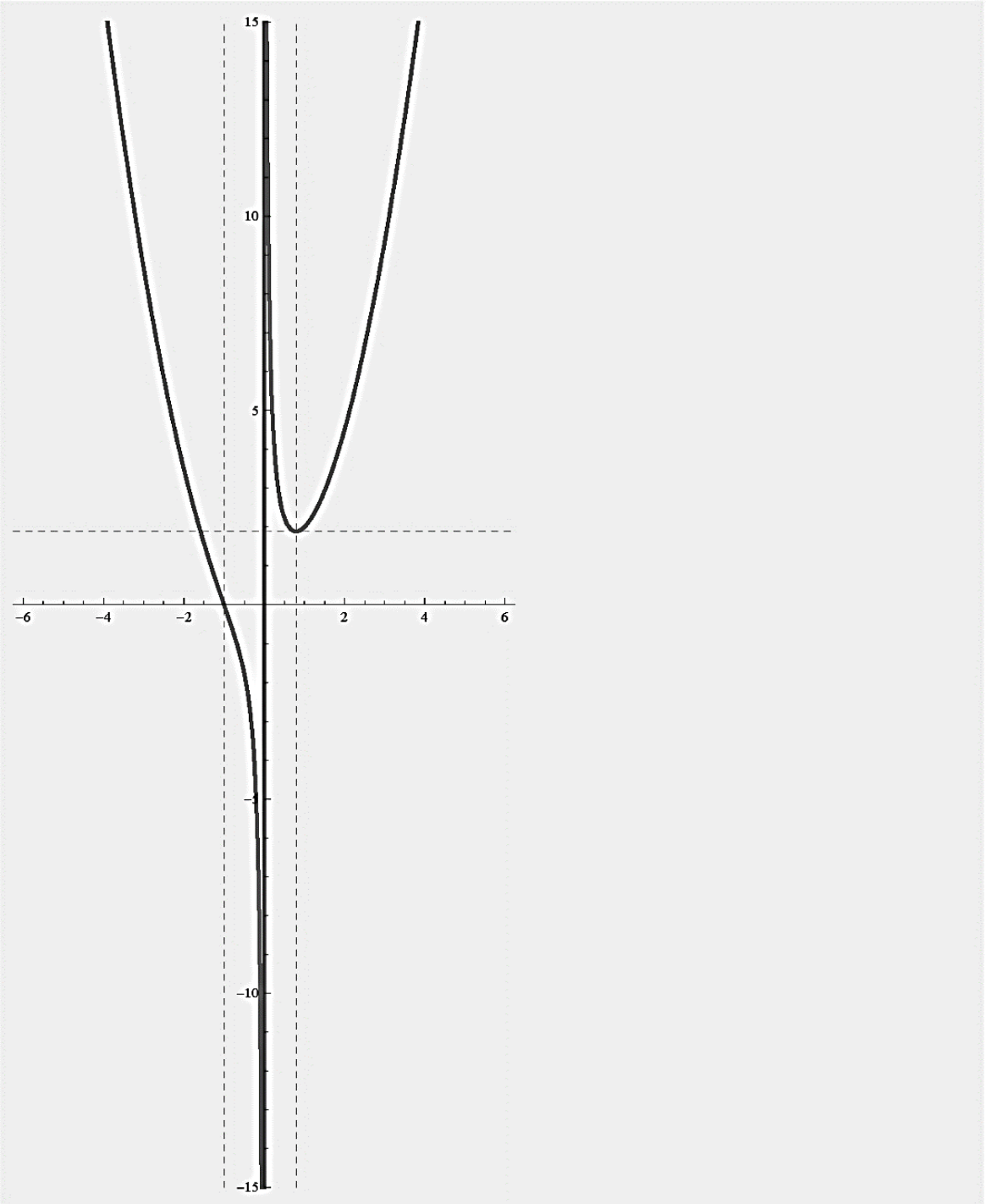
```
b2 = Limit[F[x] - k2 * x, x → -∞]
```

```
∞
```

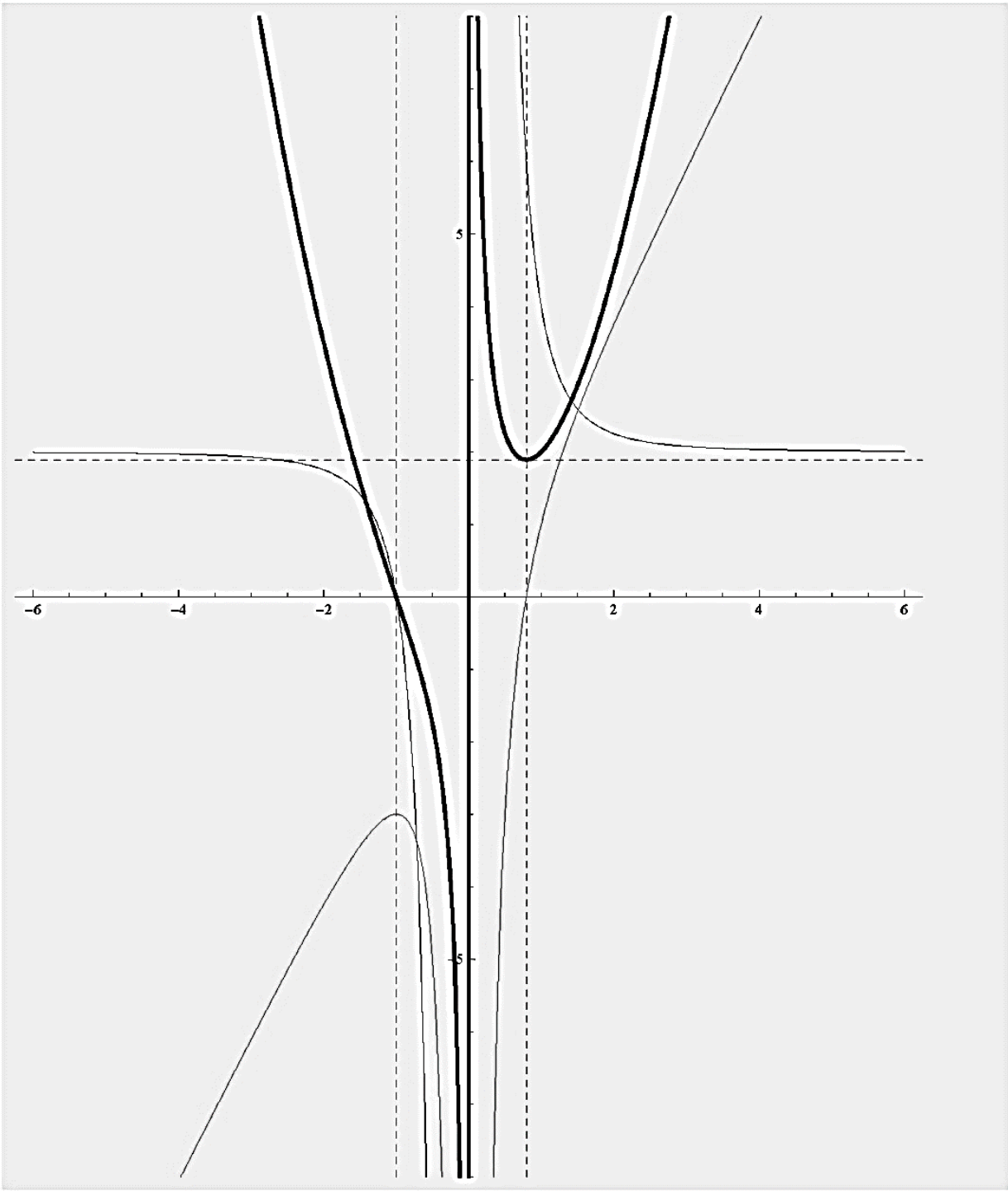
Поскольку все полученные пределы не являются конечными величинами, мы можем утверждать, что наклонных, а также горизонтальных (для случая k=0) асимптот нет.

9. Построение графика функции.

```
Plot[F[x], {x, -6, 6}, PlotStyle → {Thick, Red},
 PlotRange → {-15, 15}, AspectRatio → Automatic,
 GridLines → {{{-1, Dashed}, {0, Dashed}, {2^(-1/3), Dashed}},
   {{0, Dashed}, {1/2^(2/3) + 2^(1/3), Dashed}}}]
```



```
Plot[{F[x], G[x], H[x]}, {x, -6, 6},
 PlotStyle → {{Thick, Red}, {Thin, Green}, {Thin, Blue}},
 PlotRange → {-8, 8}, AspectRatio → Automatic,
 GridLines → {{{-1, Dashed}, {0, Dashed}, {2^(-1 / 3), Dashed}},
   {{0, Dashed}, {1/2^(2/3) + 2^(1/3), Dashed}}}]
```



В. Задания для самостоятельного выполнения

Проведите исследование функции в соответствии с номером предложенного вам варианта.

Алгоритм исследования.

1. Указать область определения функции.
2. Исследовать функцию на четность/нечетность/периодичность.
3. Найти точки пересечения графика функции с осями координат.
4. Определить промежутки знакопостоянства функции.
5. Найти производную, область ее определения и критические точки для функции.
6. Найти промежутки возрастания/убывания, точки экстремума и экстремумы.
7. Определить промежутки выпуклости и точки перегиба.
8. Исследовать поведение функции на бесконечности и в окрестностях точек разрыва.
9. Построить график функции.

Варианты заданий:

1. $f(x) = \frac{1}{1+x^2}$
2. $f(x) = \ln(1+x^2)$
3. $f(x) = \frac{2x}{1+x^2}$
4. $f(x) = \ln\frac{1+x}{1-x}$
5. $f(x) = \frac{1}{1-x^2}$
6. $f(x) = \ln\frac{1}{x^2}$
7. $f(x) = \frac{x}{1-x^2}$
8. $f(x) = \ln(1+e^x)$
9. $f(x) = \frac{(x+1)(x-2)}{(x-1)(x+2)}$
10. $f(x) = \ln((x-1)(x-2)^2(x-3)^3)$
11. $f(x) = \frac{1}{1+x} - \frac{2}{x^2} + \frac{1}{1-x}$
12. $f(x) = \log_x 2$
13. $f(x) = \frac{1}{1-2^{x/(1-x)}}$
14. $f(x) = 2 + \sqrt{1-x}$
15. $f(x) = 1 - e^{-x}$
16. $f(x) = \sqrt{1-x^3}$
17. $f(x) = x^3 - 3x + 2$
18. $f(x) = \frac{x^3}{(1-x)(1+x)^2}$
19. $f(x) = 3 + 2\cos 3x$
20. $f(x) = e^x \cos x$
21. $f(x) = \ln(\sin x)$
22. $f(x) = \cos(\ln x)$

Тема 2. Языки программирования высокого уровня. Основные операторы и типы данных. Консольный ввод/вывод.

А. Теоретическая справка

Исторический экскурс

Язык программирования — это формальная знаковая система, определяющая набор лексических, синтаксических и семантических правил, предназначенная для записи программ, содержащих действия, которые выполнит исполнитель (компьютер) под ее управлением.

Первые языки программирования возникали еще до появления современных электронных вычислительных машин.

Некоторые историки признают, что первым в мире программистом была Ада Лавлейс, дочь Байрона, которая в период с 1840 по 1843 гг. детально описала метод нахождения чисел Бернулли (фактически создала программный код) для предлагаемой Чарльзом Бэббиджем аналитической машины. Однако ввиду недостаточной развитости технологий того времени, аналитическая машина так и не была построена. (Позднее, в конце XX века, именем Ады Лавлейс был назван один из языков программирования — Ada).

В 30-40-х годах XX века были разработаны математические абстракции, такие, как лямбда-исчисление, машина Тьюринга и др. для формализации алгоритмов.

Программистам, работавшим на первых действующих ЭВМ в 1940-1950-х годах, трудно позавидовать: они писали программы, как правило, непосредственно в машинном коде в виде нулей и единиц. Эту форму записи принято считать языком программирования первого поколения. При этом в разных ЭВМ использовались различные коды, что требовало изменения программы при переносе ее на другую машину.

Языки программирования второго поколения также были машинно-зависимыми, однако позволяли упростить написание программ за счет введения символьных обозначений машинных команд. Это позволило облегчить

восприятие программ для человека. Такие языки, как правило, называются языками ассемблера. Однако, при использовании ассемблера становится необходимым перевод программы на язык машинных кодов, для чего были написаны специальные автоматизированные системы перевода, получившие название ассемблеров.

Языки первого и второго поколения принято называть низкоуровневыми.

С середины 1950-х годов стали появляться языки третьего поколения. К их числу относят такие языки как Фортран, Лисп и Кобол. Эти языки были универсальными и уже не имели жесткой зависимости от аппаратной платформы, поскольку обладали такой чертой, как абстракция. В них были введены смысловые конструкции, кратко описывающие структуры данных и операции над ними. Такие языки программирования называют языками высокого уровня. Зависимость от платформы перекладывается на инструментальные программы — трансляторы, компилирующие (переводящие) текст, написанный на языке высокого уровня, в элементарные машинные команды (инструкции). Поэтому, для каждой платформы разрабатывается платформенно-уникальный транслятор для каждого высокоуровневого языка.

Такого рода оторванность высокоуровневых языков от аппаратной реализации компьютера помимо множества плюсов имеет и минусы. В частности, она не позволяет создавать простые и точные инструкции к используемому оборудованию. Программы, написанные на языках высокого уровня, проще для понимания программистом, но менее эффективны, чем их аналоги, создаваемые при помощи низкоуровневых языков.

В период 1960–1970-х годов были разработаны основные парадигмы языков программирования, используемые и в настоящее время.

В 1980-е годы наступил период, который можно условно назвать временем консолидации. Важной тенденцией, которая наблюдалась в разработке языков программирования для крупномасштабных систем, было сосредоточение на применении модулей — объемных единиц организации кода. Одним из направлений работ становятся визуальные (графические) языки

программирования, в которых процесс «написания» программы как текста заменяется на процесс «рисования» (конструирования программы в виде диаграммы) на экране компьютера. Такой подход обеспечивает наглядность и лучшее восприятие логики программы человеком.

В 1990-х годах в связи с активным развитием сети Интернет распространение получили языки, позволяющие создавать сценарии для веб-страниц.

В настоящее время развитие языков программирования идет в направлении повышения безопасности и надежности, создания новых форм модульной организации кода и интеграции с базами данных.

Язык программирования высокого уровня «Си»

Язык Си (англ. C) был разработан Деннисом Ритчи в период с 1969 по 1973 гг. Си создавался с одной важной целью: сделать более простым написание больших программ с минимумом ошибок по правилам процедурного программирования. При этом также требовалось, чтобы перевод в машинный код осуществлялся без добавления лишних накладных расходов для компилятора. Реализация языка оказалась удачной. Благодаря высокой скорости выполнения программ, написанных на Си, этот язык получил широкое распространение при создании как системного, так и прикладного программного обеспечения, предназначенного для решения широкого круга задач. Тем самым язык Си оказал существенное влияние на развитие индустрии программного обеспечения, а его синтаксис лег в основу для таких современных языков программирования, как C++, C#, Java и других.

Язык Си обладает следующими особенностями:

- он имеет простую языковую базу, из которой вынесены в библиотеки[1] многие существенные возможности, вроде математических функций или функций управления файлами;

[1] Библиотека в программировании — это сборник подпрограмм или объектов, используемых для разработки программного обеспечения.

- ориентирован на процедурное программирование[1], обеспечивающее удобство применения структурного стиля программирования;
- содержит систему типов, предохраняющую от бессмысленных операций;
- использует препроцессор для таких операций, как, например, определения макросов и включения файлов с исходным кодом;
- обеспечивает возможность непосредственного доступа к памяти компьютера через использование указателей;
- содержит минимальное число ключевых слов;
- предоставляет возможность передачи параметров в функцию по значению и по указателю;
- обеспечивает области действия имен (области видимости);
- позволяет создавать структуры и объединения — определяемые пользователем составные типы данных, которыми можно манипулировать как одним целым.

Однако, при написании программы на этом языке следует иметь в виду, что некоторые его элементы потенциально опасны. К сожалению, особенность языка Си состоит в том, что многие случаи неправильного использования опасных элементов не могут быть обнаружены ни при компиляции, ни во время исполнения программы, что часто приводит к ее непредсказуемому поведению. Иногда в результате неграмотного использования элементов языка появляются уязвимости в системе безопасности. Поэтому при написании программ необходимо проявлять изрядное внимание.

[1] Процедурное программирование — программирование на императивном языке (такой язык описывает процесс вычисления в виде инструкций, изменяющих состояние данных), при котором последовательно выполняемые операторы можно собрать в подпрограммы, то есть более крупные целостные единицы кода, с помощью механизмов самого языка. При этом выполнение программы сводится к последовательному выполнению операторов с целью преобразования исходного состояния памяти, то есть значений исходных данных, в заключительное, то есть в результаты.

В языке Си используются (являются допустимыми) все символы латинского алфавита A-Z, a-z, цифры 0-9 и специальные символы: запятая, точка, точка с запятой; а также знаки: +, -, *, ^, & (амперсанд), =, ~ (тильда), !, / (слеш), <, >, (,), {, }, [,], |, %, ?, ' (апостроф), " (кавычки), : (двоеточие), _ (знак подчеркивания). В последних версиях некоторых компиляторов (например, в среде Visual Studio 2013) к допустимым символам добавлены также буквы кириллического алфавита.

Из допустимых символов формируются лексемы — предопределенные константы, идентификаторы и знаки операций. В свою очередь, лексемы являются частью выражений, а из выражений составляются инструкции и операторы.

Также имеется символ #, который не может быть частью никакой лексемы, и используется в препроцессоре.

Допустимый идентификатор (лексема) — это слово, составленное из допустимых символов алфавита языка программирования, не являющееся знаком некоторой операции или разделителем. Идентификаторами являются имена, которые даются программным объектам — (именованным) константам, переменным, типам и функциям.

Некоторые лексемы являются зарезервированными словами, и поэтому не могут быть использованы в программе в качестве идентификаторов программных объектов.

Для введения в программе на Си именованных констант используется директива препроцессора **`#define`**:

```
#define имя_константы [значение]
```

Введенная таким образом константа будет действовать всюду, начиная с момента задания константы и до конца программного кода или до тех пор, пока действие заданной константы не отменено другой директивой:

```
#undefine имя_константы
```

При определении именованной константы происходит автоматическая подстановка ее значения в программном коде всюду, где употреблено имя константы.

Если для именованной константы указано некоторое значение, то для константы определяется так же и тип, соответствующий виду задаваемого значения. Различают следующие типы констант:

- числовые (целочисленные или вещественные);
- символьные (выделяются знаком апострофа);
- строковые (выделяются знаком двойных кавычек).

Ключевые слова — это лексемы, которые зарезервированы компилятором для обозначения типов переменных, класса хранения, элементов операторов.

Таблица 9.

Основные ключевые слова языка Си

Ключевое слово	**Описание**
sizeof	Операция получения размера объекта.
typedef	Описание прототипа объекта.
auto, register	Обозначение класса хранения переменных.
extern	Обозначение того, что объект описывается в другом месте (является внешним).
static	Обозначение статического объекта.
char, short, int, long, signed, unsigned, float, double, void	Обозначение типа переменных.
struct, enum, union	Обозначение специальных (сложных) типов данных.
do, for, while	Обозначение операторов цикла.
if, else	Обозначение условного оператора.
switch, case, default	Обозначение оператора выбора.
break, continue	Операторы прерывания исполнения кода.

return	Обозначение оператора возврата из функции.
inline	Обозначение того, что функция является встраиваемой (ее тело вставляется в места вызовов).
new	Динамическое выделение памяти (появилось в языке C++).
delete	Динамическое освобождение памяти (появилось в языке C++).

Операция — это некоторое действие, которое выполняется над данными — операндами, посредством которого может быть вычислено значение — результат выполнения операции. Каждой операции в Си соответствует свой знак.

Операнд — это константа, переменная, выражение или вызов какой-либо определенной в программе функции.

Операции различают по количеству участвующих в ней операндов. Существуют операции унарные (с одним операндом), бинарные (с двумя операндами) и тернарные (с тремя операндами).

Совокупность операций и операндов образуют выражения.

Выражение — это упорядоченный набор операций над данными. У каждого выражения имеется значение — результат выполнения всех операций, входящих в выражение. Порядок выполнения операций в выражении зависит от формы записи и от приоритета выполнения операций. При вычислении значения выражения могут изменяться значения некоторых, входящих в его состав переменных или производиться неявное преобразование типов в выражении. Среди выражений выделяют класс лево-допустимых выражений (L-value) — выражений, которые могут присутствовать слева от знака присваивания (=).

Таблица 10.

Список основных операций в языке Си и их приоритет

Знак	**Описание операции**	**Тип**	**Приоритет**
( )	Вызов функции или подвыражение f(…)	Выражение	1
[ ]	Выделение элемента массива a[k]	Выражение	
.	Прямой выбор	Выражение	
->	Опосредованный выбор	Выражение	
++	Префиксный[1] инкремент (увеличение значения переменной на единицу)	Унарный	
- -	Префиксный декремент (уменьшение значения переменной на единицу)	Унарный	
!	Логическое отрицание	Унарный	2
&	Получение адреса переменной	Унарный	
*	Получение значения по адресу (разыменование указателя)	Унарный	
(тип)	Преобразование к указанному в скобках типу	Унарный	
-	Изменение знака числа	Унарный	2
+	Плюс (в дополнение к унарному минусу), определяет знак числа (не является обязательным)	Унарный	
~	Поразрядное отрицание	Унарный	
sizeof	Вычисление размера	Унарный	
new	Динамическое выделение памяти	Унарный	
delete	Динамическое освобождение памяти	Унарный	

[1] Префиксный оператор ставится перед операндом

Знак	Описание операции	Тип	Приоритет
*	Умножение	Бинарный	3
/	Деление	Бинарный	
%	Остаток от деления	Бинарный	
+	Сложение	Бинарный	4
-	Вычитание	Бинарный	
<<	Поразрядный сдвиг влево (или запись в поток в C++)	Бинарный	5
>>	Поразрядный сдвиг вправо (или чтение из потока в C++)	Бинарный	
<	Меньше	Бинарный	6
<=	Меньше или равно	Бинарный	
>	Больше	Бинарный	
>=	Больше или равно	Бинарный	
= =	Равно	Бинарный	7
!=	Не равно	Бинарный	
&	Поразрядное логическое И	Бинарный	8
^	Поразрядное исключающее ИЛИ	Бинарный	9
|	Поразрядное логическое ИЛИ	Бинарный	10
&&	Логическое И	Бинарный	11
||	Логическое ИЛИ	Бинарный	12
() ? :	Условная операция	Тернарный	13
=	Присваивание	Бинарный	14
+=	Составное сложение	Бинарный	
-=	Составное вычитание	Бинарный	
*=	Составное умножение	Бинарный	
/=	Составное деление	Бинарный	

Знак	Описание операции	Тип	Приоритет
%=	Составное получение остатка от деления	Бинарный	14
<<=	Составной поразрядный сдвиг влево	Бинарный	
>>=	Составной поразрядный сдвиг вправо	Бинарный	
&=	Составное поразрядное логическое И	Бинарный	
|=	Составное поразрядное логическое ИЛИ	Бинарный	
^=	Составное поразрядное исключающее ИЛИ	Бинарный	
,	Перечисление	Бинарный	15
++	Постфиксный[1] инкремент	Унарный	
--	Постфиксный декремент	Унарный	

ОБРАТИТЕ ВНИМАНИЕ! НЕКОТОРЫЕ ЗНАКИ ОПЕРАЦИЙ МОГУТ ИСПОЛЬЗОВАТЬСЯ С РАЗНЫМ ЧИСЛОМ ОПЕРАНДОВ, НАПРИМЕР, МИНУС МОЖЕТ БЫТЬ КАК УНАРНОЙ, ТАК И БИНАРНОЙ ОПЕРАЦИЕЙ.

Операторы в Си предназначены для осуществления действий и для управления ходом выполнения программы.

Для построения простейших программ, реализующих выполнение линейных алгоритмов, требуются три вида операторов:

Вид оператора	Описание
;	Пустой оператор. Не совершает никаких действий и может находиться в любом месте программы.
выражение ;	Инструкция. Некоторое элементарное действие.
{…}	Блок вычислений или составной оператор. Каждый блок вычислений определяет область видимости объявленных в нем переменных.

Другие операторы языка Си будут рассмотрены в следующих темах.

[1] Постфиксный (суффиксный) оператор ставится после операнда.

Структура программы, написанной на языке Си

Инструкции программы, написанной на языке Си, могут находиться в файле с расширением .c (для языка C) или .cpp (для языка C++). Данный файл, по своей сути, является текстовым файлом, содержимое которого преобразуется компилятором в исполняемый (машинный) код, помещаемый в .exe-файл.

Как правило, в начале программы ставятся команды препроцессора, необходимые для подключения к вашей программе файлов с описанием используемых функций языка, объявления констант и глобальных переменных, а также функций. Все операторы, используемые в программе могут находиться только внутри каких-либо функций.

Функция — это самостоятельная единица программы, которая спроектирована для реализации конкретной подзадачи[1].

Выполнение каждой функции не зависит от ее расположения в тексте программы, а начинается только после того, как она будет вызвана из какой-либо другой функции. При этом выполнение программы всегда начинается с функции с именем **`main()`**, поэтому она должна в обязательном порядке присутствовать в любой программе в одном экземпляре.

Рассмотрим пример простой программы, которая выводит на экран предложение «Hello, World!»

```
1  #include <cstdio>
2  #include <conio.h>
3
4  void main(){
5       printf("Hello, World!");
6       _getch();
7  }
```

В этом примере, в 4 строке, объявляется основная функция программы. Ее описание состоит из указания типа возвращаемого значения, имени и круглых скобок — обязательного признака любой функции. Далее в фигурных скобках содержится так называемое тело функции, в котором находятся выполняемые ею операторы.

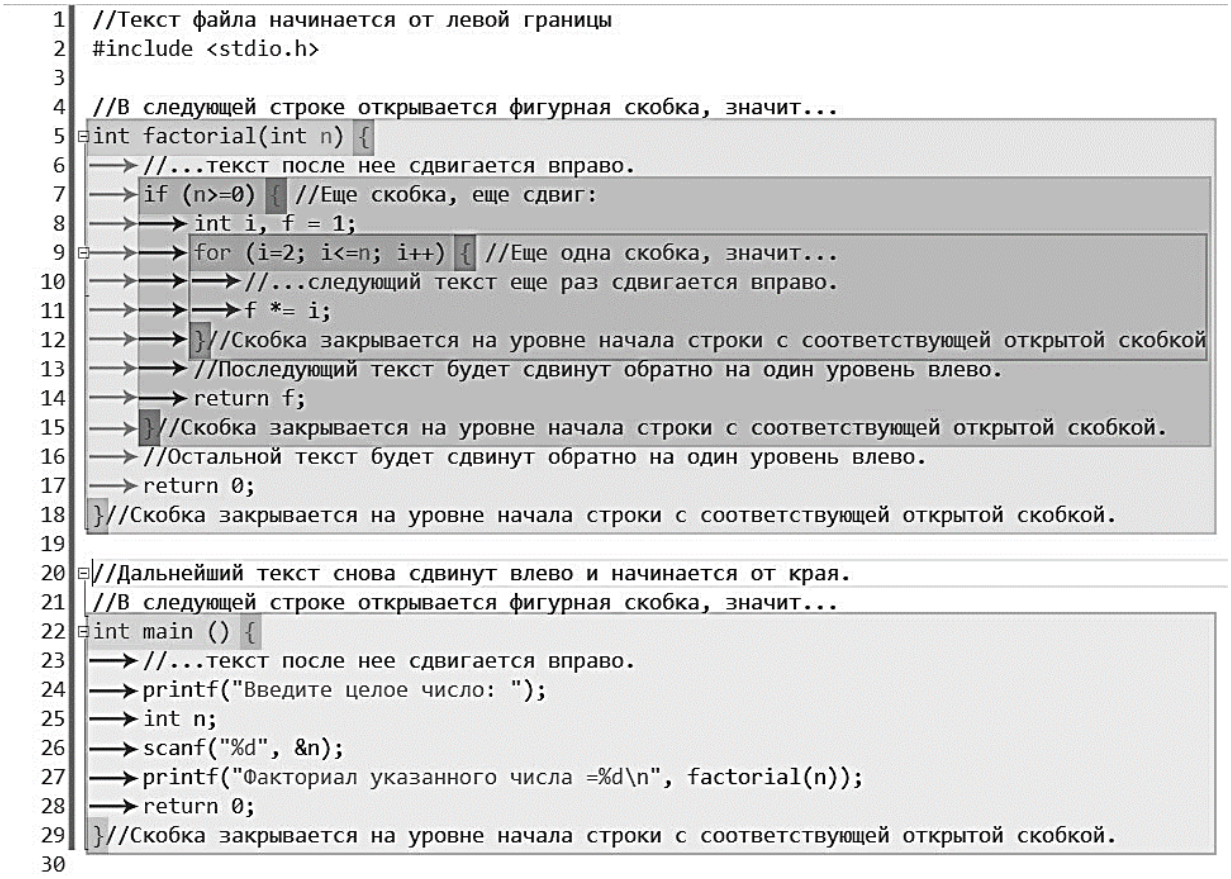
[1] Более подробно о функциях речь пойдет в теме 3.

В 5 строке содержится команда вывода информации на экран — вызов стандартной функции **`printf()`**. В ее скобках задан **параметр** — в данном случае это текст, который будет выведен на экран.

Как только программа завершает свою работу (выполняет все операторы), ее окно закрывается. Поэтому, чтобы можно было успеть увидеть отображенную в нем информацию, необходимо дождаться реакции пользователя, например, нажатия какой-либо клавиши на клавиатуре. Этого можно добиться при помощи вызова операции **`_getch()`**. Она приостанавливает работу программы до тех пор, пока не будет нажата какая-либо клавиша на клавиатуре.

Для того, чтобы можно было использовать (вызывать) в программе некоторую функцию, ее описание обязательно должно содержаться в тексте программы. Компилятору необходима информация о типе возвращаемого функцией значения, имени и списка параметров. Поскольку мы воспользовались уже существующими функциями **`printf()`** и **`_getch()`**, нам нужно указать, где находятся их описания. Для этого мы включили в наш код строки (1 и 2), содержащие инструкции препроцессора, позволяющие подключить необходимые файлы. Файл с именем "cstdio" содержит описание для **`printf()`**, а файл "conio.h" — для **`_getch()`**.

Обратите внимание! При написании программ на языке Си или Си++, а также большинстве других языков, очень важно придерживаться правил выравнивания текста. Это сильно облегчает восприятие программы. Основным правилом здесь является то, что после каждой открывающейся фигурной скобки текст в последующих строках следует сдвигать вправо на одну позицию табуляции. Закрывающаяся фигурная скобка должна возвращать уровень текста на одну позицию табуляции влево (см. рис. 2).



```
1  //Текст файла начинается от левой границы
2  #include <stdio.h>
3
4  //В следующей строке открывается фигурная скобка, значит...
5  int factorial(int n) {
6      //...текст после нее сдвигается вправо.
7      if (n>=0) { //Еще скобка, еще сдвиг:
8          int i, f = 1;
9          for (i=2; i<=n; i++) { //Еще одна скобка, значит...
10             //...следующий текст еще раз сдвигается вправо.
11             f *= i;
12         }//Скобка закрывается на уровне начала строки с соответствующей открытой скобкой
13         //Последующий текст будет сдвинут обратно на один уровень влево.
14         return f;
15     }//Скобка закрывается на уровне начала строки с соответствующей открытой скобкой.
16     //Остальной текст будет сдвинут обратно на один уровень влево.
17     return 0;
18 }//Скобка закрывается на уровне начала строки с соответствующей открытой скобкой.
19
20 //Дальнейший текст снова сдвинут влево и начинается от края.
21 //В следующей строке открывается фигурная скобка, значит...
22 int main () {
23     //...текст после нее сдвигается вправо.
24     printf("Введите целое число: ");
25     int n;
26     scanf("%d", &n);
27     printf("Факториал указанного числа =%d\n", factorial(n));
28     return 0;
29 }//Скобка закрывается на уровне начала строки с соответствующей открытой скобкой.
30
```

Рис. 2. Блочная структура программы.

Поскольку подавляющее большинство программ оперирует значительно бóльшим, чем в предыдущем примере, количеством данных, для их хранения требуется использовать переменные, а также в именованные константы.

Переменная — поименованная область памяти, адрес которой можно использовать для осуществления доступа к данным (как для чтения, так и для изменения).

Константа — поименованная или иным способом адресованная область памяти, изменение содержания которой рассматриваемой программой не предполагается или запрещается.

Каждая константа и переменная в языке Си обладает определенным типом данных. Он необходим для того, чтобы определить набор допустимых операций, диапазон возможных значений данных из набора и способ их хранения в памяти.

Таблица 11.

Основные типы данных в языке Си

Тип данных	**Размер (бит)**	**Диапазон значений**	**Описание**
bool	8	false, true	Булевский тип. Данные такого типа могут принимать значение «Истина» (true) или «Ложь» (false). Для совместимости типов языке Си все значения, отличные от нуля, трактуются как истина, а 0 — как ложь.
char	8	-128…127	Малые целые числа. При выводе на экран интерпретируются в качестве символов. Хранимое числовое значение соответствует коду символа в ASCII-таблице символов.
unsigned char	8	0…255	Малые целые числа без знака и коды символов.
enum	16	-32 768…32 768	Упорядоченные поименованные наборы целых значений
short	16	-32 768…32 768	Короткое целое число
unsigned short	16	0…65 535	Беззнаковое короткое целое число
int, long	32	-2 147 483 648… …2 147 483 647	Целые числа
unsigned int	32	0… 4 294 967 296	Целые числа без знака

Тип данных	Размер (бит)	Диапазон значений	Описание
long long	64	- 9 223 372 036 854 775 808 … 9 223 372 036 854 775 807	Длинные целые числа
unsigned long long	64	0… 18 446 744 073 709 551 616	Длинные целые числа без знака
float	32	±3.4E-38 … ±3.4E+38	Вещественные числа
double	64	±1.7E-308… …±1.7E+308	Вещественные числа удвоенной точности
long double	80	±3.4E-4932… …±1.1E+4932	Длинные вещественные числа

Перед использованием всякая переменная должна быть объявлена (описана). Для объявления переменной необходимо указать ее тип и имя. Заглавные и строчные буквы в языке Си различаются.

Примеры описания переменных и констант:

```
1   int i;//Описание целочисленной переменной
2   unsigned int a;//Описание беззнаковой целочисленной переменной
3   short x = 10; /* Описание короткой целой переменной с
4   одновременным присвоением ей значения 10 */
5   float t = 3.4; /* Объявление вещественной переменной и присвоение ей
6   значения 3.4*/
7   char c = 'ю';/* Объявление символьной переменной с присвоением ей значения
8   символьной константы */
9   bool a, b = false;/*Объявление пустой булевской переменной a и булевской
10  переменной b, которая принимает значение «ложь»*/
11
12  //Объявление константы строкового типа со значением «Привет»:
13  #define s "Привет"
14  /*В языке Си символьные и строковые константы различаются по типу кавычек*/
15
16  int s = 11;//Ошибка! Это имя в программе уже занято константой
```

Обратите внимание! В программах могут присутствовать комментарии. Их содержимое игнорируется компилятором, и поэтому там может содержаться любой текст. Комментариями являются символы идущие после двойной косой черты (//) до конца строки или все символы, заключенные между знаками /* … */. Комментарии используются для внесения в программу пометок для программиста.

Представление данных в памяти компьютера

Для представления информации в памяти компьютера (как числовой так и не числовой) используется двоичный способ кодирования. Элементарная ячейка памяти ЭВМ имеет длину 8 бит (1 байт). Каждый байт имеет свой номер (его называют адресом). Наибольшую последовательность бит, которую ЭВМ может обрабатывать как единое целое, называют машинным словом. Длина машинного слова зависит от разрядности процессора и может быть равной 32, 64 битам и т.д.

Именно двоичным представлением данных обусловлены и те границы диапазонов значений, которые были представлены выше, в таблице 11. Рассмотрим для примера тип unsigned char, на который в памяти отводится 8 бит. В таком типе можно хранить до $2^8 = 256$ различных значений (от 00000000 (0) до 11111111 (255)).

Для представления беззнаковых чисел используется так называемый прямой код. Перевод чисел из прямого двоичного кода в десятичное представление осуществляется достаточно просто при помощи разложения исходного числа по степеням основания системы счисления. Так, например, значение $01001011_2 = 0*2^7+1*2^6+0*2^5+0*2^4+1*2^3+0*2^2+1*2^1+1*2^0 = 64_{10}+8_{10}+2_{10}+1_{10} = 75_{10}$.

Для типа char также используется 8 бит, однако старший бит[1] в этом случае определяет знак числа. Для хранения абсолютного значения остается только 7 бит. Таким образом, появляются числа от (0)0000000 до (0)1111111 — неотрицательные значения и от (1)0000000 до (1)1111111 — отрицательные значения.

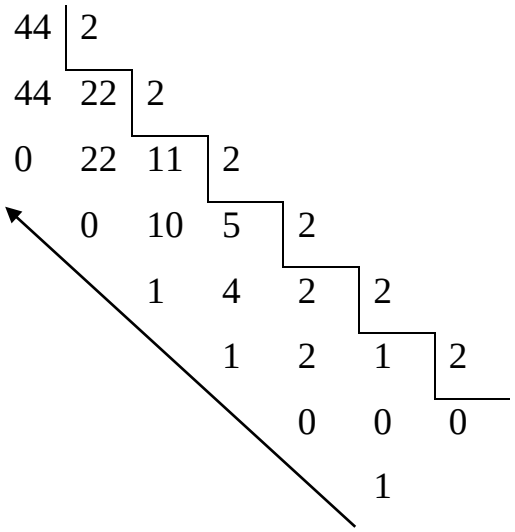
Перевод значений положительного диапазона из десятичной системы счисления в двоичную не представляет трудностей. Однако, такой алгоритм приведет к проблемам при применении к отрицательному диапазону значений, поскольку в этом случае появится число –0 (10000000).

[1] Самая левая двоичная цифра в числе называется старшим битом.

Для того, чтобы устранить этот недостаток, числа представляются в так так называемом дополнительном коде. Дополнительный код целого положительного числа совпадает с его прямым кодом, а для целого отрицательного числа может быть получен по следующему алгоритму:

- записать прямой код модуля числа;
- инвертировать его (заменить единицы нулями, нули — единицами);
- прибавить к инверсному коду единицу.

Возьмем для примера число -44. Переводим в двоичную систему счисления модуль числа:



Прямой код модуля числа находится путем добавления ведущих нулей к двоичной записи числа до отводимого в памяти количества бит (для типа char — 8): 00101100. Инвертированный код: 11010011. После прибавления единицы получим: 11010100 — дополнительный код числа -44.

Обратное преобразование выполняется, соответственно, по обратному алгоритму. Возьмем для примера число 10110101_2. Вычитаем единицу: 10110100, инвертируем: 01001011, преобразуем к десятичной системе счисления: $2^6+2^3+2^1+2^0=75$.

Данные в переменных типов bool, short, int, long long, а также их беззнаковых вариантах хранятся аналогично. Нужно лишь учитывать, что одно и то же число будет занимать разный объем памяти:

Переменная	Представление значения в памяти
bool x=true;	00000001
char c = 'я'; //Код = -1	11111111
unsigned char u='я';//Код = 255	11111111
short s = 13;	0000000000001101
int i =13;	00000000000000000000000000001101
int j = -13;	11111111111111111111111111110011
long long k=13;	0000000000000000000000000000000000000000000000000000000000001101
long long l=-13;	1111111111111111111111111111111111111111111111111111111111110011

Представление вещественных чисел в памяти несколько отличается от представления целочисленных значений. Вещественное число представляется в виде $\mathbf{M \times 10^P}$, где M — мантисса — дробная величина, находящаяся в диапазоне $1 \leq M < 10$, а P — порядок. Например, число 3445,21 в таком виде будет записано как $3{,}44521 \times 10^3$, а число 0,043122 — как $4{,}3122 \times 10^{-2}$. На экране компьютера запись таких чисел обычно выглядит следующим образом: 3.44521E3 и 4.3122E-2 соответственно.

Алгоритм получения двоичного представления числа в памяти в типе float будет следующим:

1) модуль вещественного числа переводится в двоичную систему счисления[1];
2) двоичное число нормализуется, т.е. записывается в виде $M \times 2^p$, где M — мантисса (ее целая часть всегда равна 1_2 и поэтому из записи числа исключается), а p — порядок, записанный в десятичной системе счисления;

[1] Целая часть числа переводится при помощи деления числа на 2 и сбору остатков от деления в порядке, обратном получению. Дробная часть получается умножением дробной части на 2 и сбором целых частей в порядке получения. (Сами целые части при этом отбрасываются). Так, например, для числа 0,3125 получим: 0,3125*2 = 0,625; 0,625*2 = 1,25; 0,25*2 = 0,5; 0,5*2 = 1,0.

Таким образом, $0{,}3125_{10} = 0{,}0101_2$.

3) к порядку добавляется смещение (+127 для типа float и +1023 для типа double)[1], а затем смещенный порядок переводится в двоичную систему счисления;
4) учитывая знак заданного числа (0 — положительное; 1 — отрицательное), формируется его представление в памяти компьютера. Для типа float запись содержит 32 бита. Старший бит отвечает за знак числа, следующие 8 бит — за порядок, оставшиеся 23 бита — за мантиссу. Для типа double представление будет аналогичным, с той лишь разницей, что на всё число отводится 64 бита, из которых старший также отвечает за знак числа, следующие 11 — за порядок, а оставшиеся 52 — за мантиссу.

В качестве примера рассмотрим число -17,078125. Следуя описанному выше алгоритму:

1) переводим модуль числа в двоичную систему счисления $17{,}078125_{10} = 10001{,}000101_2$;
2) нормализуем число: $1{,}0001000101 * 2^4$;
3) добавляем к порядку смещение для float: 4+127 = 131; для double — 4+1023=1027 и переводим порядок в двоичную систему счисления, получая 10000011 для float и 10000000011 для double;
4) записываем представление числа в памяти

в типе float:

1	10000011	00010001010000000000000
знак	порядок	мантисса

[1] Порядок равный 00000000_2 соответствует минимальному возможному значению порядка. Всего с помощью 8 бит, отводимых на порядок в типе float, можно закодировать 256 различных значений от -128 до 127. Некоторые из них зарезервированы для хранения служебных значений, таких как NaN (не число) или INF (бесконечность). Поэтому фактически наименьшим является значение p=-127, кодируемое как 00000000_2. Соответственно, порядок p=0 будет кодироваться числом $127=01111111_2$ (это и есть величина смещения). Аналогичные рассуждения можно провести и для типа double.

и в типе double:

1	10000000011	0001000101000000000000000000000000000000000000000000000000
знак	порядок	мантисса

Организация ввода/вывода данных (стиль «Си»)

Для вывода данных на экран в языке Си используется функция форматированного вывода **`printf()`**, с которой мы уже встречались ранее. Однако ее можно использовать и для вывода данных в формате отличном от строкового. Описание функции (ее прототип) имеет следующий вид:

```
int printf (format-string [, argument...]);
```

Здесь format-string — строка, задающая формат вывода данных, за которой может следовать произвольное количество аргументов. Строка формата может содержать произвольный текст и флаги. На места флагов будут подставлены данные из последующих аргументов. Каждый флаг начинается с символа «**%**», а количество и порядок аргументов должны соответствовать числу и порядку флагов.

Таблица 12.

Флаги, используемые в строке формата

Флаг	**Примечание**
%d, %i	Целое десятичное число со знаком.
%f	Вещественное число (float).
%lf	Вещественное число удвоенной точности (double).
%6.2f	Вещественное число, на которое отводится 6 знаков, включая 2 после запятой.
%e	Вещественное число в научной нотации (например, значение 135,675 будет записано в этом формате как 1.35675e+002. Значение, расположенное после буквы «e» означает необходимость умножения на 10 в указанной степени.

%g	Автоматический выбор формы %e или %f, в зависимости от того, какой будет записан короче.
%o	Вывод восьмеричного числа.
%u	Целочисленное значение без знака.
%lld	Число типа long long (длинное целое).
%x	Шестнадцатеричное целое значение без знака, буквы нижнего регистра.
%X	Шестнадцатеричное целое значение без знака, буквы верхнего регистра.
%c	Символьное значение (char).
%s	Строковое значение.
%p	Значение указателя.
%%	Выводит символ %, не требует наличия дополнительных аргументов.

Также в строке формата могут присутствовать специальные символы:

Таблица 13.

Символ	**Описание**
\n	Символ конца строки (переход на новую строку).
\r	Возврат каретки.
\t	Табуляция (сдвиг вправо на несколько символов).
\\	Знак косой черты (обратный слеш).
\"	Кавычка.

Отметим также, что для вывода на экран кириллических символов необходимо добавить в программу поддержку русского языка. Для этого необходимо в самом начале функции **`main()`** написать:

```
1  SetConsoleOutputCP(1251);
2  SetConsoleCP(1251);
```

Описание этих функций находится в файле "Windows.h", который нужно подключить к программе при помощи директивы **`#include`**.

Пример вывода значений переменных разных типов:

```
1   #include <Windows.h>
2   #include <cstdio>
3   #include <conio.h>
4
5   void main(){
6         SetConsoleCP(1251);
7         SetConsoleOutputCP(1251);
8         int i = 16;
9         char c = '@';
10        float x = 3.5;
11        printf("Заданы переменные: i=%d, c=%c, x=%f\n", i, c, x);
12        printf("Символ '%c' имеет код %i.\n", c, c);
13        printf("Число i=%d в десятичной, =%o в восьмеричной "
14                "и =%X в шестнадцатеричной системе.\n", i, i, i);
15        printf("Вещественное число в разных формах записи:"
16               " %f = %.3f = %g\n", x, x, x);
17        _getch();
18  }
```

В результате на экране будет выведено:

```
Заданы переменные: i=16, c=@, x=3.500000
Символ '@' имеет код 64.
Число i=16 в десятичной, =20 в восьмеричной и =10 в шестнадцатеричной
системе.
Вещественное число в разных формах записи: 3.500000 = 3.500 = 3.5
```

Для ввода данных с клавиатуры используется функция **`scanf_s()`**. Ее описание (прототип) имеет следующий вид:

```
int scanf_s (format-string [, argument...]);
```

Можно заметить, что функция чтения данных с клавиатуры очень похожа на ее аналог для вывода данных. Первый параметр — строка формата может иметь те же конструкции, что были описаны для **`printf()`**, однако при вводе данных существуют свои особенности.

Прежде всего, необходимо учесть, что в качестве дополнительных аргументов следует указывать не сами переменные, а их адреса. Например, если необходимо ввести с клавиатуры целое число, запись будет выглядеть так:

```
1   int i;
2   scanf_s("%d", &i);
```

Обратите внимание, что перед названием переменной появляется знак «**&**».

Функция **`scanf_s()`** возвращает количество реально прочитанных и присвоенных переменным значений.

Кроме того, при указании флагов в строке формата можно использовать некоторые дополнительные команды.

Значок звездочки после процента и перед названием флага (например, **%*d**) позволяет считать данные, но не помещать их в переменную (если при чтении множества значений с клавиатуры какие-то из них необходимо пропустить).

Например, пользователь вводит с клавиатуры три числа через пробел «10 20 30» и эта последовательность считывается командой:

```
1  int x, y;
2  scanf_s("%d%*d%d", &x, &y);
```

Тогда переменная **x** станет рана 10, а **y** — 30. Значение 20 будет считано, но не будет присвоено ни одной переменной.

Указание произвольных символов в строке формата приведет к их отбрасыванию из исходных данных[1]. Так, если пользователь введет время в формате «15:30», то отдельно часы и минуты можно будет получить при помощи команды:

```
1  int m, s;
2  scanf_s("%d:%d", &m, &s);
```

При этом двоеточие будет отброшено, поскольку оно указано в строке формата. Однако, если пользователь введет данные в виде «15.30», то это приведет к ошибке чтения, поскольку функция **scanf_s** попытается прочитать точку как целое число.

Ввод/вывод данных в стиле языка C++

Язык C++, ставший расширением языка C, добавил некоторые новые возможности для ввода/вывода информации. Для использования его возможностей необходимо написать в начале программы:

```
1  #include <iostream>
2  using namespace std;
```

[1] Пробелы, символы табуляции и конца строки всегда отбрасываются автоматически, если только не осуществляется чтение отдельных символов (флаг «%c»).

Затем, для вывода информации можно будет использовать объект **cout**, а для чтения — **cin**. Способ их применения показан ниже в сравнении со стилем языка С.

Стиль С

```
1  #include <cstdio>
2  #include <conio.h>
3  #include <Windows.h>
4  void main(){
5    SetConsoleCP(1251);
6    SetConsoleOutputCP(1251);
7    int a, b;
8    printf ("Введите целое a:");
9    scanf_s("%d", &a);
10   printf("a=%d\n", a);
11   printf("Введите 2 числа:");
12   scnaf_s("%d %d", &a, &b);
13   print("a+b=%d\n", a+b);
14   _getch();
15 }
```

Стиль Си++

```
1  #include <iostream>
2  using namespace std;
3
4  void main(){
5    setlocale(LC_ALL, ".1251");
6
7    int a, b;
8    cout << "Введите целое a:";
9    cin >> a;
10   cout << "a=" << a << endl;
11   cout << "Введите 2 числа:";
12   cin >> a >> b;
13   cout <<"a+b="<<a+b<<"\n";
14   system("pause");
15 }
```

Как можно заметить из приведенного примера, при использовании стиля С++ нет необходимости заботиться об указании типов считываемых и выводимых на экран значений.

Обратите внимание! Для русификации программ в стиле С++ используется функция **setlocale(LC_ALL, ".1251")**, а для приостановки программы перед закрытием — функция **system("pause")**.

Обратите внимание! В дальнейшем в разделе теоретической справки, как правило, будут приводиться неполные фрагменты программ. Чтобы они заработали, необходимо будет самостоятельно составить правильную структуру программы. Так, например, для простых примеров из темы 2 можно будет использовать следующую структуру:

```
1  #include <iostream>
2  using namespace std;
3
4  void main(){
5       setlocale(LC_ALL, ".1251");
6
7       //Здесь разместить код фрагмента
8
9       system("pause");
10 }
```

Основные видов алгоритмов

Как известно из школьного курса информатики, алгоритмы подразделяются на три основных вида:

- линейные,
- разветвляющиеся,
- циклические.

В линейных алгоритмах, которые мы рассматривали до сих пор, все команды выполняются строго последовательно, друг за другом. Разветвляющийся алгоритм — это алгоритм, который содержит команду ветвления, то есть составную команду, в которой та или иная серия команд выполняется после проверки условия. Циклические алгоритмы содержат команду повторения — составную команду, в которой часть действий (тело цикла) выполняется несколько раз.

Организация разветвляющихся алгоритмов

Оператор ветвления (условный оператор) — оператор, обеспечивающий выполнение различных команд в зависимости от выполнения некоторого условия (истинности или ложности логического выражения).

В языке программирования Си существует 3 возможности для создания разветвляющихся алгоритмов.

1. Использование условной операции, общая запись которой выглядит следующим образом:

```
(условие)?операция_1:операция_2;
```

Она позволяет вычислить одно из двух выражений в зависимости от истинности или ложности определенного условия.

Пример:

```
1  int a = -2;
2  int b = (a>=0) ? a : -a;/*Условная операция, реализующая функцию
3  вычисления модуля числа*/
```

2. Применение условного оператора «if». Синтаксис оператора может включать как две ветви (полный условный оператор), так и одну ветвь (сокращенный условный оператор). Полный условный оператор, в отличие от

сокращенного, содержит действие или список действий, которые будут выполняться в случае ложности указанного в скобках после слова «if» условия. Альтернативная ветвь программы указывается в этом случае после ключевого слова «else».

Общий вид оператора можно записать следующим образом:

```
if (условие) {
      действия1; //выполняются, если условие истинно
} else {
      действия2; //выполняются, если условие ложно
}
```

Здесь условие — это либо булевская переменная, либо выражение, возвращающее булевское значение, либо переменная или выражение результат которого может быть приведен к булевскому типу.

Обратите внимание! В языке Си любое целое (или приводимое к целому) значение отличное от 0 трактуется как истинное, а нулевое значение — как ложное.

Пример[1]:

```
1   float a = 1., b=-3., c = 2.;
2   float d = b*b – 4*a*c;
3   float x1, x2;
4   if (d>=0){                                          Сокращенный условный оператор
5         x1 = (-b+sqrt(d))/(2*a);                      Полный условный оператор
6         x2 = (-b-sqrt(d))/(2*a);
7         cout << x1 << " " << x2 << endl;
8   }
9   else {
10        cout << "Невозможно определить корни" << endl;
11  }
```

Обратите внимание! Если при выполнении или невыполнении условия необходимо выполнить несколько операторов, они должны быть заключены в фигурные скобки. В противном случае (выполняется только одно действие), наличие фигурных скобок не является обязательным.

Условия, указываемые в скобках, могут быть простыми, например, **`d >= 0`**, или составными. Составные условия состоят из нескольких простых, объединенных логическими операциями «И» (обозначается одним или

---

[1] В примере используется функция вычисления квадратного корня — **`sqrt`**, для работы которой требуется подключить в начале программы файл cmath инструкцией **`#include <cmath>`**.

двумя знаками амперсанда «**&&**»), «ИЛИ» (обозначается одной или двумя вертикальными чертами «**||**»), «НЕ» (обозначается восклицательным знаком «**!**»). Пример составного условия: **d >= 1 || d <= -1** (будет истинным, или когда значение переменной **d** больше 1, или когда оно меньше -1).

Обратите внимание! Количество знаков логической операции («&» и «|») определяет полноту проводимых логических вычислений. Так, в случае использования конструкции «**X & Y**», будут в обязательном порядке вычислены значения выражений **X** и **Y**. Однако в условии вида «**X && Y**» значение выражения **Y** будет вычисляться только тогда, когда значение **X** — истинно, ведь в противном случае результат не зависит от **Y** и будет ложным.

---

3. Использование оператора выбора (**switch**). Конструкция оператора имеет несколько (две или более) ветвей. Его удобно использовать в ситуациях, когда существует необходимость в выполнении различного набора действий при многих различных значениях некоторой переменной (например, выводить название сезона в соответствии с номером месяца). Принципиальным отличием этой инструкции от условного оператора является то, что выражение, определяющее выбор исполняемой ветви, имеет не логическое, а целое значение, либо значение, тип которого может быть приведен к целому.

Общий вид оператора выбора может быть записан следующим образом:

```
switch(переменная){
case <значение_1>: {
      действия_1;
      [break;]
}
case <значение_2>: {
      действия_2;
      [break;]
}
…
case <значение_n>: {
      действия_n;
      [break;]
}
[default: {
      действия_по_умолчанию;
}]
}
```

Обратите внимание! В конце каждой ветви может находиться необязательный оператор **break**. Его использование прерывает выполнение оператора выбора после выполнения действий в соответствующей ветке. При отсутствии оператора **break,** будут выполнены все дальнейшие действия, даже если они находятся в других ветвях.

Пример:

```
1   int m;
2   cout << "Введите номер месяца: ";
3   cin >> m;
4   switch (m){
5         case 12:
6         case 1:
7         case 2: {
8               cout << "Это зимний месяц\n";
9               break;
10        }
11        case 3:
12        case 4:
13        case 5: {
14              cout << "Это весенний месяц\n";
15              break;
16              }
17        case 6:
18        case 7:
19        case 8: {
20              cout << "Это летний месяц\n";
21              break;
22              }
23        case 9:
24        case 10:
25        case 11: {
26              cout << "Это осенний месяц\n";
27              break;
28        }
29        default: {
30              cout << "Неверный номер месяца\n";
31        }
32  }
```

Организация циклических алгоритмов

Для организации многократного повторения части алгоритма в языке Си могут быть использованы циклы трех видов:

- цикл с параметром (**for)**;
- цикл с предусловием (**while**);
- цикл с постусловием (**do…while**).

Цикл с параметром используется в тех случаях, когда заранее известно, сколько именно повторений однотипных действий (итераций) должно быть совершено в цикле.

Формат записи оператора цикла **for** следующий:

```
for (инициализация; условие_продолжения; изменение_счетчика){
     тело цикла;
}
```

Секция инициализации в цикле служит для объявления и/или присвоения начального значения ранее объявленной переменной-счетчику, которая отвечает за выполнение условия в цикле.

Обратите внимание! Если переменная-счетчик была объявлена в цикле, по завершении цикла эта переменная будет уничтожена.

Условие продолжения цикла бывает, как правило, связано с переменной-счетчиком и определяет количество итераций цикла.

Раздел изменения счетчика позволяет изменять значение соответствующей переменной.

Тело цикла может содержать один или несколько операторов. Если в теле содержится только один оператор, фигурные скобки можно опустить.

Обратите внимание! Любая из перечисленных четырех секций цикла с параметром может отсутствовать. Так, если переменная-счетчик описана и инициализирована выше, может отсутствовать секция инициализации. Если в цикле не задано условие продолжения, цикл будет выполняться бесконечно. При отсутствии раздела изменения счетчика, это должно происходить в теле цикла, иначе он, скорее всего, также будет повторяться бесконечно много раз.

Следует помнить о порядке выполнения секций цикла. Так, инициализация выполняется в первую очередь при заходе в цикл. Затем проверяется условие, и если оно истинно, то осуществляются действия, перечисленные в теле цикла. В последнюю очередь выполняется секция изменения значения счетчика, затем снова проверяется условие. Если оно остается истинным, то снова выполняется тело цикла и т.д.

Ниже приведен фрагмент программы, содержащий пример цикла, выполняющего подсчет суммы первых **N** натуральных чисел:

```
1   #define N 10
2   int s = 0;
3   for (int i = 1; i<=N; i++){
4        s += i;
5   }
6   cout << s << endl; // 55
```

Циклы с предусловием и постусловием применяются в тех случаях, когда заранее неизвестно, сколько раз должны быть выполнены операторы в теле цикла. Отличие двух этих видов лишь в том, что у первого сначала проверяется условие, и затем, если оно истинно, выполняется тело цикла, а у второго — сначала выполняются действия в теле цикла и только потом проверяется условие продолжения. Таким образом, цикл с предусловием может и не начаться, если условие окажется сразу ложным, а, вот, действия в цикле с постусловием выполнятся обязательно хотя бы один раз.

Формат записи цикла с предусловием:

```
while (условие_продолжения_цикла){
     тело цикла;
}
```

Формат записи цикла с постусловием:

```
do {
     тело цикла;
} while (условие_продолжения_цикла);
```

ОБРАТИТЕ ВНИМАНИЕ! В СЛУЧАЕ, КОГДА В ТЕЛЕ ЦИКЛА НАХОДИТСЯ ЛИШЬ ОДИН ОПЕРАТОР, ФИГУРНЫЕ СКОБКИ МОГУТ ОТСУТСТВОВАТЬ.

В следующем примере, демонстрирующем применение цикла с предусловием, на экран выводится последовательность чисел Фибоначчи[1], значение максимального из которых не превосходит заданной пользователем величины:

```
1   int a = 1;//Переменная будет содержать предыдущее число Фибоначчи
2   int b = 1;//Переменная будет содержать текущее число Фибоначчи
3   int max;//Переменная для хранения максимального значения в последовательности
4   cout << "Введите максимальное значение числа Фибоначчи: ";
```

[1] Последовательность Фибоначчи начинается с двух единиц, а все последующие числа вычисляются как сумма двух предыдущих.

```
5   cin >> max;//Определение предела последовательности
6   cout << a << " ";//Вывод первого числа последовательности (1)
7   while (b <= max){
8         int c = a+b;//Вычисление следующего числа Фибоначчи
9         a = b;//Текущее число Фибоначчи становится предыдущим
10        b = c;//Следующее число Фибоначчи становится текущим
11        cout << a << " ";//Выводим очередное значение на экран
12  }
```

В следующем фрагменте программы находится сумма вводимых с клавиатуры элементов последовательности ненулевых чисел. Ввод данных останавливается, если вводится нулевое значение:

```
1   int x;
2   int s = 0;
3   do {
4         cin >> x;//Ввод числа (осуществляется в любом случае до проверки условия)
5         s += x; //Добавление числа к сумме
6   } while (x != 0);//Продолжаем, пока пользователь не введет 0
7   cout << "Сумма элементов последовательности =" << s;
```

Операторы управления циклом

В некоторых ситуациях бывает удобно использовать операторы, позволяющие принудительно завершить цикл или перейти на следующую его итерацию без выполнения части тела цикла.

Оператор **`break`** используется для принудительного прерывания текущего цикла.

В следующем фрагменте программы определяется, является ли введенное с клавиатуры число простым:

```
1   bool prost = true;//позволяет определить, поделилось ли число на что-то,
2   //кроме себя и 1
3   int n;//исследуемое число
4   cout << "Введите число для проверки: ";
5   cin >> n;
6   for (int i = 2; i<=n/2; i++){
7         if (n%i==0){//Делим исследуемое число по порядку на все числа
8                     //начиная от 2 до его половины. Если оно на что-то поделится
9                     //(остаток от деления на какое-то число равен 0)…
10              prost = false;//… значит оно не простое
11              break;//Дальнейшая проверка бессмысленна, поскольку ясно, что число
12                    //уже не простое, а значит цикл можно остановить
13        }
14  }
15  cout<<"Это число"<<((prost)?" ":" не ")<<"простое"<<endl;
```

Оператор **`continue`** используется для принудительного перехода на следующую итерацию цикла.

Следующий код вычисляет и выводит на экран значение суммы:

$$S = \sum_{\substack{i=-5, \\ i\neq 0}}^{5} \frac{1}{i}.$$

```
1   int s = 0;//Переменная для хранения суммы.
2   for (int i = -5; i <=5; i++){// i изменяется от -5 до 5.
3         if (i==0)//Если i равно 0,
4               continue;  //то сразу переходим на следующую итерацию.
5         s += 1 / i;//Сюда попадаем только для i не равных 0.
6   }
7   cout << "s=" << s;//Выводим результат
```

Б. Примеры программ

1. Конвертер температур из градусов по шкале Фаренгейта в градусы по шкале Цельсия. Применяемая формула преобразования: $C° = \frac{5}{9}(F° - 32)$.

```
1   #include <cstdio>
2   #include <conio.h>
3
4   //Конвертер температур
5   void main(){
6         //Преобразование градусов по шкале Фаренгейта
7         //в шкалу Цельсия.
8         float f, c;//Объявление переменных
9         printf_s("F=");//Вывод подсказки
10        scanf_s("%f", &f);//Ввод данных с клавиатуры
11        //Преобразование:
12        c=5/9*(f - 32);  // ОШИБКА!!! Производятся целочисленные вычисления
13        printf_s("C=%f\n", c);  // C = 0.000000 (при F=100)
14        c=5./9*(f - 32); // Правильный вариант записи формулы
15        printf_s("C=%f\n", c);  // C = 37.777779 (при F=100)
16        _getch();//Пауза для просмотра результата
17  }
```

Обратите внимание! При вычислении значения выражения в строке 12, первым выполняется действие 5/9. Поскольку оба операнда здесь — целые числа, то их частное также трактуется как целое число. Таким образом, 5/9=0. Для получения вещественнозначного частного, следует один из операндов привести к вещественному типу, например, как это сделано в строке 14.

2. Дан номер года. Требуется определить количество дней в этом году. При написании программы следует учесть, что високосными являются года,

номера которых делятся на 4, однако не делятся на 100. Исключением являются года, номера которых также делятся на 400. Таким образом, 1900 год не является високосным, а 2000 — является.

```
#include <cstdio>
#include <conio.h>

//Определение количества дней в году.
void main(){
	int y;//Номер года
	int days = 365;//Число дней в году.
	printf_s("Year=");
	scanf_s("%d", &y);//Вводим данные с клавиатуры
	//Определяем високосный год или нет.
	if (y % 4 == 0 //Номер кратен 4
		&& //И
		(y % 100 != 0 //... не кратен 100
		|| //ИЛИ
		y % 400 == 0) //... кратен 400
		){
		days++;//Число дней в високосном году.
	}
	printf_s("Days: %d", days);
	_getch();
}
```

3. Даны три числа — длины сторон треугольника. Написать программу, которая проверяет, существует ли такой треугольник и, если существует, выводит его площадь.

Для программы воспользуемся тем свойством, что сумма длин любых двух сторон треугольника должна быть больше третьей стороны.

```
#include <cstdio>
#include <conio.h>
#include <cmath>
#include <Windows.h>

//Проверка существования треугольника и нахождение его площади
void main(){
	SetConsoleOutputCP(1251);
	float a, b, c;//Длины сторон треугольника
	printf_s("a=");
	scanf_s("%f", &a);
	printf_s("b=");
	scanf_s("%f", &b);
	printf_s("c=");
	scanf_s("%f", &c);
	//Проверяем, что сумма длин любых двух сторон больше третьей
	if ((a + b) > c &&
		(b + c) > a &&
```

```
19              (a + c) > b){
20              printf_s("Треугольник существует\n");
21              float p = (a + b + c)/2;//Находим полупериметр
22              //Площадь находим по формуле Герона:
23              float s = sqrt(p*(p - a)*(p - b)*(p - c));
24              printf_s("Площадь=%g", s);
25        }
26        else {
27              printf_s("Такого треугольника не существует!\n");
28        }
29        _getch();
30  }
```

4. Арифметические действия над числами пронумерованы следующим образом: 1 – сложение, 2 – вычитание, 3 – умножение, 4 – деление. Дан номер действия N и два вещественных числа A и B. Выполнить над числами указанную операцию и вывести результат на экран.

```
1   include <cstdio>
2   #include <conio.h>
3   #include <Windows.h>
4
5   //Выполнение арифметических действий над числами
6   void main(){
7         SetConsoleOutputCP(1251);//Поддержка кириллицы
8         int N;//Номер, кодирующий операцию
9         float A, B;//Числа-операнды
10        printf_s("Введите номер операции: ");
11        scanf_s("%d", &N);
12        printf_s("Введите первый операнд: ");
13        scanf_s("%f", &A);
14        printf_s("Введите второй операнд: ");
15        scanf_s("%f", &B);
16        float r;//Результат
17        char op;//Символ применяемой операции (будем использовать при выводе)
18        switch (N){//Проверяем, какая операция должна выполняться
19        case 1:{
20              r = A + B;
21              op = '+';
22              break;
23        }
24        case 2:{
25              r = A - B;
26              op = '-';
27              break;
28        }
29        case 3:{
30              r = A * B;
31              op = '*';
32              break;
33        }
34        case 4:{
35              r = A / B;  //В случае, когда B=0 результатом будет являться
```

```
36              //1.#INF или -1.#INF, означающий + или – бесконечность соотв-но.
37              op = '/';
38              break;
39        }
40        default: {
41              r = NAN;//Если введен неверный номер операции,
42              //то в результате получаем неопределенность -1.#IND
43              op = '?';//При этом символ операции также неопределен
44        }
45        }
46        printf_s("A %c B = %g", op, r);
47        _getch();
48  }
```

5. Дано число N. Вывести последовательно первых N чисел Фибоначчи.

```
1   #include <iostream>
2   using namespace std;
3
4   void main(){
5         setlocale(LC_ALL, ".1251");
6         int N;
7         cout << "N=";
8         cin >> N;//Вводим количество числе Фибоначчи для вывода
9         int f1 = 1, f2 = 1, f3 = 1;//Начальные значения последовательности
10        cout << "1: " << f1 << endl;//Первое число Фибоначчи
11        cout << "2: " << f2 << endl;//Второе число Фибоначчи
12        for (int i = 3; i <= N; i++){
13              //В цикле вычисляются третье и последующие числа Фибоначчи
14              f1 = f2;//Изменяем предыдущее число Фибоначчи
15              f2 = f3;//Изменяем текущее число Фибоначчи
16              f3 = f1 + f2;//Вычисляем следующее число Фибоначчи
17              cout << i << ": " << f3 << endl;
18        }
19        system("pause");
20  }
```

6. Даны два целых числа A и B. Найти их наибольший общий делитель.

```
1   #include <iostream>
2   using namespace std;
3
4   //Нахождение наибольшего общего делителя
5   void main(){
6         setlocale(LC_ALL, ".1251");
7         int a, b;
8         cout << "Введите числа A и B через пробел или Enter: ";
9         cin >> a >> b;
10        //Для нахождения НОД получаем остаток от деления
11        //большего числа на меньшее, до тех пор пока
12        //одно из чисел не станет равным нулю.
13        //Тогда последний делитель будет являться НОД-ом.
14        do {
15              if (a > b) a %= b;
```

```
16              else b %= a;
17          } while (a*b != 0);//Любое из чисел может быть равным нулю
18          //После окончания цикла одно из чисел = 0, а другое - искомое значение
19          //Можем вывести их сумму в качестве результата:
20          cout << "НОД=" << a + b << endl;
21          system("pause");
22      }
```

В. Задания для самостоятельного выполнения

1. Написать программу для преобразования температуры из шкалы Цельсия в шкалу Фаренгейта и в шкалу Кельвина.
2. Написать программу для прямой и обратной конвертации дюймов в сантиметры, миль в километры, граммов в караты, пудов в килограммы, километры/час в метры/секунду, градусы в радианы.
3. Дано целое число. Вывести его строку-описание вида «Положительное четное число», «Нуль», «Отрицательное нечетное число» и т.п.
4. Дано целое число в диапазоне 1-999. Вывести его описание вида «Четное двузначное число», «Нечетное трехзначное число» и т.д.
5. Дана дата рождения некоторого человека в пределах с XIX по XXI век. Определить знак его зодиака по восточному и западному календарям.
6. Дано целое положительное число N. Вывести N-е число Фибоначчи.
7. Дано целое положительное число N. Вывести ближайшее к нему число Фибоначчи.
8. Написать программу, вычисляющую значение наибольшего общего делителя для N данных целых чисел.
9. Написать программу, вычисляющую значение наименьшего общего кратного:

   а) для 2 чисел;

   б) для N чисел.

10. Дано число N, равное сумме цифр страниц некоторой книги. Определить, сколько страниц содержит такая книга или вывести сообщение, что такой книги не существует. Например, для N=46 число страниц = 10, для N=48

число страниц = 11, при этом не существует книги, сумма цифр страниц которой будет равна 47.

11. Даны целые положительные числа N и K. Найти значение выражения: $1^K+2^K+3^K+\ldots+N^K$. Чтобы избежать целочисленного переполнения, вычисления проводить в переменной типа double.

Тема 3. Основы процедурного программирования. Функции. Шаблоны функций. Разделение кода по файлам.

А. Теоретическая справка

Разработка собственных дополнительных функций

Функция — это поименованный фрагмент программы (подпрограмма), к которому, при необходимости, можно обратиться из другого места программы. Для выполнения действий, указанных в функции ее необходимо вызвать[1]. После выполнения подпрограммы, управление возвращается обратно, к месту ее вызова.

Функция может принимать параметры (переменные, через которые можно передавать данные) и возвращать некоторое значение.

Функция должна быть соответствующим образом объявлена и определена. В объявлении, кроме имени, должен содержаться список типов и имен передаваемых параметров, а также, тип возвращаемого значения, например:

```
1   float f(int x);
```

Здесь **`float`** — это тип возвращаемого значения, **`f`** — имя функции, **`int x`** — ее целочисленный параметр.

```
1   void g(int a, float x);
```

Во втором примере, функция с именем **`g`** принимает два параметра (целочисленный и вещественный), но никаких значений в точку вызова не возвращает, о чем свидетельствует ее тип (**`void`**).

Определение функции содержит исполняемый ею код (тело функции).

[1] Единственная функция, которая не вызывается в программе — это **`main()`**. Она запускается автоматически операционной системой при старте программы.

ОБРАТИТЕ ВНИМАНИЕ! ЕСЛИ ТИП ФУНКЦИИ ОТЛИЧЕН ОТ **`void`**, ТО В ЕЕ ТЕЛЕ ДОЛЖНО ПРИСУТСТВОВАТЬ КЛЮЧЕВОЕ СЛОВО **`return`**, ПОСЛЕ КОТОРОГО СЛЕДУЕТ ЗНАЧЕНИЕ ИЛИ ВЫРАЖЕНИЕ, СОДЕРЖАЩЕЕ ВОЗВРАЩАЕМЫЙ РЕЗУЛЬТАТ.

Для того, чтобы использовать ранее определенную функцию, необходимо в требуемом месте программного кода указать ее имя и перечислить передаваемые параметры.

Функция определяет собственную (локальную) область видимости, куда входят ее параметры, а, также, переменные, объявленные непосредственно в ее теле. Такие переменные будут недоступны за пределами их области видимости.

Рассмотрим программу, в которой создана функция, вычисляющая квадрат числа, передаваемого ей в качестве параметра.

```
1   #include <iostream>
2   using namespace std;
3   /*Функция возведения вещественного числа в квадрат.
4    * Она имеет один вещественный параметр x; возвращает вещественное
5    * значение — квадрат числа x.
6    */
7   float kvd(float x){
8         float r = x*x;
9         return r;
10  }
11  void main(){
12        setlocale (LC_ALL, ".1251");
13        float a;
14        cout << "Введите вещественное число: ";
15        cin >> a;
16        float y = kvd(a);//Вызов описанной ранее функции.
17        cout << "a^2 = " << y;
18        system("pause");
19  }
```

ОБРАТИТЕ ВНИМАНИЕ! ФУНКЦИЯ НЕ МОЖЕТ БЫТЬ ОПИСАНА ВНУТРИ ДРУГОЙ ФУНКЦИИ. ПРИ ЭТОМ ЕЕ ОПИСАНИЕ ОБЯЗАТЕЛЬНО ДОЛЖНО РАСПОЛАГАТЬСЯ В ТЕКСТЕ ПРОГРАММЫ ДО ПЕРВОГО ВЫЗОВА.

Рассмотрим, как будет работать такая программа. Сразу же после запуска, управление получает основная функция — **`main()`** (начинает выполняться первый ее оператор, находящийся в 12 строке). Далее, до 16 строки действия выполняются подряд. В 16 строке, справа от знака присваивания,

находится вызов функции возведения числа в квадрат — **kvd()**[1]. Параметры, которые располагаются в скобках рядом с ее именем (в данном случае — это «**a**») называются фактическими, поскольку содержат конкретные значения. Параметры, указанные в описании функции, используются в области ее видимости и называются формальными (поскольку при описании их значения еще не известны). После вызова функции, ей передается управление и начинает выполняться код в 8 строке. Перед этим значения фактических параметров копируются в формальные. В восьмой строке вычисляется значение квадрата параметра **x**, а полученный при этом результат возвращается в точку вызова и там попадает в переменную **y**, объявленную в 16 строке.

Ниже приведена условная схема изменения состояния памяти компьютера при работе программы.

Строка 15. Пользователь вводит значение переменной a.

	Состояние памяти								
Адрес[2]	AF00	AF04	AF08	AF0C	AF10	AF14	AF18	AF1C	AF20
Значение	2.0[3]								
Переменная	a								
Область[4]	main								

Строка 16. Вызывается функция kvd. Значение фактического параметра копируется в формальный, осуществляется переход к строке 7.

	Состояние памяти								
Адрес	AF00	AF04	AF08	AF0C	AF10	AF14	AF18	AF1C	AF20
Значение	2.0				2.0				
Переменная	a				x				
Область	main				kvd				

[1] Описание расположено в 7 строке.

[2] Здесь и далее указываются условные значения адресов памяти. При этом в нашем примере адреса идут с шагом в 4 для краткости записи, поскольку все используемые переменные занимают по 4 байта в памяти.

[3] Значение взято для примера. Пользователь может ввести любое вещественное число с клавиатуры.

[4] Здесь и далее имеется в виду область видимости переменной (например, имя функции, в которой она определена).

Строка 8. Вычисляется квадрат числа, его значение записывается в переменную **r**.

	Состояние памяти								
Адрес	AF00	AF04	AF08	AF0C	AF10	AF14	AF18	AF1C	AF20
Значение	2.0				2.0	4.0			
Переменная	a				x	r			
Область	main				kvd	kvd			

Строка 9. Осуществляется выход из функции и возврат к строке 16. Значения локальных переменных при этом уничтожаются. Функция **kvd** возвращает результат в функцию **main**, который попадает там в переменную **y**.

	Состояние памяти								
Адрес	AF00	AF04	AF08	AF0C	AF10	AF14	AF18	AF1C	AF20
Значение	2.0	4.0							
Переменная	a	y							
Область	main	main							

Рекурсивные функции

Рекурсивные функции — это функции, вызывающие самих себя. Рекурсия бывает прямой и косвенной. При прямой рекурсии вызов рекурсивной функции **f()** располагается непосредственно в ее собственном теле. При косвенной рекурсии вызов функции **f()** может находиться в некоторой функции **g()**, вызов которой, в свою очередь, располагается в функции **f()**.

Максимальное число рекурсивных вызовов функции без возвратов, которое происходит во время выполнения программы, называется глубиной рекурсии.

Любая рекурсивная функция обязана содержать условие остановки рекурсии, при котором она перестает вызывать саму себя. В противном случае произойдет аварийная остановка работы программы из-за переполнения стека вызовов функции.

Классический пример рекурсии — вычисление значения факториала некоторого числа n (n! = 1·2·…·n). Рассмотрим способ записи такого алгоритма. Отметим, что в его основе использован тот факт, что n! = n * (n-1)!

```
1   #include <iostream>
2   using namespace std;
3
4   //Рекурсивная функция вычисления факториала:
5   long double fct(unsigned short n){
6         if (n <= 1) {
7               //Условие остановки рекурсии
8               //0! = 1! = 1
9               return 1;
10        }
11        else {
12              // n! = n * (n-1)!
13              return n * fct(n - 1);
14        }
15  }
16
17  void main(){
18        setlocale(LC_ALL, ".1251");
19        cout << "Введите n:";
20        unsigned short n;//Факториал определен для неотрицательных чисел
21        cin >> n;
22        /* Поскольку факториал - очень быстро растущая
23         * функция, результат будем хранить в переменной,
24         * допускающей максимально большой диапазон значений.
25         */
26        long double f = fct(n);
27        cout << n << "!=" << f << endl;
28        system("pause");
29  }
```

Заголовочные файлы и распределение функций по файлам

Как было отмечено нами ранее, описание любой функции должно быть сделано до момента первого ее использования (до первого вызова). Однако это требование иногда может приводить к неразрешимым, казалось бы, ситуациям. Рассмотрим случай косвенной рекурсии.

```
1   float f(float x){
2         //…
3         g(x-1);
4         //…
5   }
6   float g(float x){
7         //…
8         f(x-1);
9         //…
10  }
```

Такое использование косвенной рекурсии будет приводить к ошибке компиляции, поскольку функция **`g(x)`** описана ниже своего вызова в функции **`f(x)`**. Если же поменять их местами, то ошибка сохранится, но уже для **`f(x)`**.

Для разрешения подобных ситуаций в программе можно произвести описание всех функций, а затем написать их реализации. Описание функции содержит ее заголовок, но вместо фигурных скобок с телом функции сразу ставится точка с запятой:

```
1   float f(float x); //Описание функции f
2   float g(float x); //Описание функции g
3
4   float f(float x){ //Реализация функции f
5         //…
6         g(x-1); //Здесь вызов осуществляется после описания, сделанного в строке 2,
7                 //поэтому ошибки не возникает.
8         //…
9   }
10  float g(float x){ //Реализация функции g
11        //…
12        f(x-1);
13        //…
14  }
```

Кроме того, текст программы может быть размещен в разных файлах. Это удобно делать в тех случаях, когда различные функции можно сгруппировать по их смысловому назначению. Например, требуется написать программу, которая решает задачу нахождения значения модуля суммы двух векторов на плоскости. Тогда все функции, выполняющие операции над векторами, следует разместить в отдельном файле (например, «vectors.cpp»). При этом оставшиеся действия, направленные на решение конкретной поставленной задачи, а также осуществляющие ввод/вывод данных, могут быть размещены в другом файле (например, «main.cpp»). Такой способ компоновки функций сделает программу более легкой для восприятия. Однако в этом случае необходимо связать друг с другом несколько файлов, чтобы функции из файла «vectors», вызываемые в файле «main» были известны компилятору (напомним, что ему должно быть известно описание каждой функции перед вызовом).

С этой целью можно было бы использовать уже известную конструкцию **`#include`**, которая позволила бы включить содержимое файла «vectors.cpp» в «main.cpp». Однако это приведет ошибке, поскольку получится, что каждая функция для работы с векторами реализована в программе

дважды: там, где написана изначально, в «vectors.cpp», но также и в файле «main.cpp», поскольку добавлена туда через директивы включения. Для решения этой проблемы в языке Си используется специальный тип файлов, куда включаются заголовки функций без их реализации, который имеет расширение «.h». Именно их можно безопасно подключать через инструкцию **`#include`**.

Рассмотрим, как можно использовать функции, выделенные в отдельный файл на следующем примере.

Пусть требуется вычислить значения корней квадратного уравнения. Для решения поставленной задачи, операции нахождения дискриминанта и квадратного корня можно выделить в отдельные функции и разместить их реализацию в файле uravn.cpp, а заголовки — в uravn.h.

Заголовочный файл («uravn.h») с описанием подключаемых функций:

```
1  //Функция вычисления дискриминанта для уравнения
2  //с коэффициентами a, b и c.
3  float diskr(float a, float b, float c);
4  //Функция вычисления первого корня для уравнения с
5  //коэффициентами a, b, c.
6  float koren1(float a, float b, float c);
7  //Функция вычисления второго корня для уравнения с
8  //коэффициентами a, b, c.
9  float koren2(float a, float b, float c);
```

Файл с реализацией функций («uravn.cpp»):

```
1  //Необходимо подключить заголовочный файл,
2  //чтобы компилятор мог проверить соответствие
3  //описаний функций их реализациям
4  #include "uravn.h"
5  //Необходимо для вычисления значения квадратного корня
6  #include "math.h"
7  //Функция вычисления дискриминанта для уравнения
8  //с коэффициентами a, b и c.
9  float diskr(float a, float b, float c){
10       return b * b - 4 * a * c;
11 }
12 //Функция вычисления первого корня для уравнения с
13 //коэффициентами a, b, c.
14 float koren1(float a, float b, float c){
15       float d = diskr(a, b, c);
16       return (-b + sqrt(d)) / (2 * a);
17       //Если d<0, функция sqrt вернёт значение NAN (-1.#IND)
18       //(«не число»).
19 }
20 //Функция вычисления второго корня для уравнения с
21 //коэффициентами a, b, c.
```

```
22  float koren2(float a, float b, float c){
23       float d = diskr(a, b, c);
24       return (-b - sqrt(d)) / (2 * a);
25  }
```

Файл с основной частью программы («main.cpp»):

```
1   //Необходимо подключить заголовочный файл,
2   //чтобы компилятор мог проверить соответствие
3   //вызова функции ее описанию
4   #include "uravn.h"
5
6   #include <iostream>
7   using namespace std;
8
9   void main(){
10       setlocale(LC_ALL, ".1251");
11       cout << "Введите коэффициенты квадратного уравнения.\n";
12       float a, b, c;
13       cout << "a=";
14       cin >> a;
15       cout << "b=";
16       cin >> b;
17       cout << "c=";
18       cin >> c;
19       cout << "Ответ:\n";
20       cout << "x1=" << koren1(a, b, c) << endl;
21       cout << "x2=" << koren2(a, b, c) << endl;
22       system("pause");
23  }
```

При необходимости мы можем воспользоваться файлами «uravn.h» и «uravn.cpp» и в другом проекте, например, для нахождении корней биквадратного уравнения. Достаточно будет скопировать файлы с функциями в папку проекта и подключить их. При этом нет необходимости вникать в код и анализировать, какие его части требуется скопировать в новый файл, как это пришлось бы делать, если все функции располагались вместе.

```
1   //Необходимо подключить заголовочный файл,
2   //чтобы компилятор мог проверить соответствие
3   //вызова функции ее описанию
4   #include "uravn.h"
5
6   #include <iostream>
7   using namespace std;
8
9   void main(){
10       setlocale(LC_ALL, ".1251");
11       cout << "Введите коэффициенты биквадратного уравнения.\n";
12       float a, b, c;
13       cout << "a=";
14       cin >> a;
15       cout << "b=";
16       cin >> b;
```

```
17        cout << "c=";
18        cin >> c;
19        cout << "Ответ:\n";
20        float y1 = koren1(a, b, c);
21        float y2 = koren2(a, b, c);
22        int i = 0;//Счётчик корней
23        if (y1 >= 0){
24             cout << "x" << ++i << "=" << sqrt(y1) << endl;
25             cout << "x" << ++i << "=" << -sqrt(y1) << endl;
26        }
27        if (y2 >= 0){
28             cout << "x" << ++i << "=" << sqrt(y2) << endl;
29             cout << "x" << ++i << "=" << -sqrt(y2) << endl;
30        }
31        system("pause");
32   }
```

Перегрузка функций

Перегрузка функций — это возможность использования в языке программирования одноименных подпрограмм. В языке Си для упрощения процесса трансляции существовало ограничение, согласно которому одновременно в программе не могло быть доступно более одной функции с одним и тем же именем. Однако, в Си++ такое ограничение было снято. При этом, для того, чтобы можно было однозначно определить, какой из вариантов функции необходимо использовать, перегружаемые функции должны иметь разное количество или типы аргументов (иметь различную сигнатуру). Перегруженная функция фактически представляет собой несколько разных функций, и выбор подходящей происходит на этапе компиляции.

Обратите внимание! Невозможно создать перегруженные функции, отличающиеся только типом возвращаемого значения, поскольку в таком случае нельзя однозначно определить, какой из вариантов подпрограммы следует вызывать.

Одним из наиболее общих случаев использования перегрузки является применение функции для получения определенного результата, исходя из различных параметров. Например, в вашей программе есть функция, которая по заданной дате должна определить день недели. Эту функцию можно пе-

регрузить таким образом, чтобы она верно возвращала результат, если ей переданы три числа – день, месяц, год, либо строковое представление даты в определенном формате, либо целочисленное значение даты типа **`time_t`**[1].

Шаблоны функций

Шаблон функции определяет общий набор операций, который будет применен к данным различных типов. Используя этот механизм, можно применять некоторые общие алгоритмы к широкому кругу данных. Как известно, многие алгоритмы логически одинаковы вне зависимости от типа данных, которыми они оперируют. Например, функция, определяющая наибольший из двух элементов, будет выполнять одни и те же действия, при проверки значений типов int, char, float, double и т.д.

При помощи создания функции-шаблона можно определить сущность алгоритма безотносительно к типу данных. После этого компилятор автоматически генерирует корректный код для того типа данных, для которого создается данная конкретная реализация функции на этапе компиляции.

ОБРАТИТЕ ВНИМАНИЕ! ДЛЯ ТОГО, ЧТОБЫ ФУНКЦИЯ БЫЛА СГЕНЕРИРОВАНА ПО ШАБЛОНУ НА ЭТАПЕ КОМПИЛЯЦИИ, НЕОБХОДИМО, ЧТОБЫ ВМЕСТЕ С ЕГО ОПИСАНИЕМ (В РАМКАХ ТОГО ЖЕ ФАЙЛА) НАХОДИЛСЯ ВЫЗОВ ШАБЛОННОЙ ФУНКЦИИ ДЛЯ КОНКРЕТНОГО ТИПА.

Функции-шаблоны создаются с использованием ключевого слова template. Общая форма функции-шаблона имеет следующий вид:

```
template <typename п_тип>
возвращаемый_тип имя_функции(список формальных параметров)
{
//тело функции
}
```

ОБРАТИТЕ ВНИМАНИЕ! В НЕКОТОРЫХ СТАРЫХ ПРОГРАММАХ ВМЕСТО КЛЮЧЕВОГО СЛОВА **`typename`** МОЖЕТ ИСПОЛЬЗОВАТЬСЯ КЛЮЧЕВОЕ СЛОВО **`class`**.

Здесь п_тип является параметром-типом, определяющий некоторый тип данных, который используется функцией. Этот параметр-тип может быть

[1] Иногда для задания даты/времени в языке Си используется целое число, равное количеству секунд, прошедших с 0:00:00 1 января 1900 года.

использован в определении функции. Однако это только формальное название типа, которое будет автоматически заменено компилятором на фактический тип данных во время создания конкретной версии функции.

Вызов шаблонной функции осуществляется стандартным образом:

```
имя_функции (список фактических параметров);
```

При этом происходит конкретизация шаблона для типов, соответствующих фактическим параметрам.

В шаблонах функций могут использоваться несколько различных параметрических типа. В этом случае они перечисляются через запятую:

```
template <typename T1, typename T2>
```

Б. Примеры программ

1. Даны вещественные числа a и b. Написать функцию, вычисляющую a^b. Для решения поставленной задачи можно применить формулу:

$$a^b = e^{b \ln a}.$$

Поскольку логарифм для $a \leq 0$ не определен для вещественных значений, функция будет возвращать значение 0, для отрицательного основания степени.

```
1   #include <iostream>
2   #include <cmath>
3   using namespace std;
4
5   //Функция возведения вещественного числа в
6   //вещественную степень.
7   double power(double a, double b){
8         //Для отрицательного основания возвращаем 0.
9         if (a <= 0) return 0;
10        return exp(b* log(a));
11  }
12
13  void main(){
14        double a, b, r;
15        cout << "a=";
16        cin >> a;
17        cout << "b=";
18        cin >> b;
19        r = power(a, b);
20        cout << "a^b=" << r << endl;
21        system("pause");
22  }
```

2. Дано целое положительное число N. Требуется написать рекурсивную функцию для разложения числа на простые множители.

```
1   #include <iostream>
2   #include <cmath>
3   using namespace std;
4
5   //Разложение числа на простые множители
6   void mnozh(int n){
7         //Пробуем поделить n на числа, не превышающие его корня.
8         //Если число не на что не поделится, то оно простое.
9         for (int i = 2; i <= sqrt(n); i++){
10              if (n%i == 0){//Если число поделилось без остатка
11                    //значит i - его простой множитель
12                    cout << i << " * "; //Выводим его на экран
13                    // и далее находим разложение на простые множители
14                    mnozh(n / i); // для частного
15                    return;//на этом данный вызов функции завершает работу.
16              }
17        }
18        //Если цикл закончился, а делитель не был найден,
19        //значит осталось простое число, которое является последним множителем
20        cout << n << endl;
21  }
22
23  void main(){
24        setlocale(LC_ALL, ".1251");
25        cout << "Введите число: ";
26        unsigned int N;
27        cin >> N;
28        cout << N << " = ";
29        mnozh(N);
30        system("pause");
31  }
```

Результат работы программы:

```
Введите число: 870269400
870269400 = 2 * 2 * 2 * 3 * 3 * 3 * 5 * 5 * 7 * 7 * 11 * 13 * 23
```

3. Написать программу, содержащую перегруженные функции для определения дня недели по дате, заданной в различных форматах.

```
1   #include <iostream>
2   using namespace std;
3   //Получение дня недели по числу, месяцу и году.
4   unsigned short getDayOfWeek(unsigned short d, unsigned short m,
5                               unsigned short y){
6         if (m <= 2) {
7               y--;
8               d += 3;
9         }
10        unsigned short dow = 1 + (d + y + y / 4 - y / 100 +
11                              y / 400 + (31 * m + 10) / 12) % 7;
12        return dow;
```

```
}

//Перегруженная функция для получения дня недели.
//Используется для дат, заданных в виде целого числа в формате yyyytmdd
unsigned short getDayOfWeek(unsigned int date){
	//Получаем в отдельных переменных значения числа, месяца и года.
	unsigned short d, m, y;
	d = date % 100;//Последние две цифры - это день
	date /= 100;//Откидываем две последние цифры
	m = date % 100;//Оставшиеся две последние цифры - номер месяца
	date /= 100;//Откидываем две последние цифры
	y = date;//Остался номер года.
	//Вызываем перегруженный вариант этой же функции для трех параметров.
	return getDayOfWeek(d, m, y);
}

void main(){
	setlocale(LC_ALL, ".1251");
	cout << getDayOfWeek(30, 12, 2012) << endl;
	cout << getDayOfWeek(29, 2, 2016) << endl;
	cout << getDayOfWeek(20200828) << endl;
	system("pause");
}
```

Результаты работы программы:

```
7 //30.12.2012 - воскресенье
1 //29.02.2016 - понедельник
5 //28.08.2020 - пятница
```

4. Даны две переменные некоторого типа. Написать шаблон функции, для поиска наибольшего из двух значений.

```
#include <iostream>
#include <cmath>
using namespace std;

template <typename T> //Объявление шаблона
T maximum(T a, T b){
	return (a > b) ? a : b;
}

void main(){
	int a, b;
	//Вводятся два целых числа:
	cout << "a=";
	cin >> a;
	cout << "b=";
	cin >> b;
	//Определяется наибольшее из них (шаблон конкретизируется для типа int):
	cout << "max(a, b)=" << maximum(a, b) << endl;
	char c, d;
	//Вводятся два символа:
	cout << "c=";
	cin >> c;
```

```
23          cout << "d=";
24          cin >> d;
25          //Определяется больший из них (шаблон конкретизируется для типа char):
26          cout << "max(c, d)=" << maximum(c, d) << endl;
27          system("pause");
28      }
```

Результат работы программы:

```
a=10
b=54
max(a, b)=54
c=я
d=в
max(c, d)=я
```

В. ЗАДАНИЯ ДЛЯ САМОСТОЯТЕЛЬНОГО ВЫПОЛНЕНИЯ

1. Даны координаты двух точек на плоскости. Написать функцию, определяющую длину отрезка с концами в этих точках.
2. Написать функцию вычисления двойного факториала.

$$(2N+1)!! = 1 \cdot 3 \cdot 5 \cdot \ldots \cdot 2N+1;$$

$$(2N)!! = 2 \cdot 4 \cdot 6 \cdot \ldots \cdot 2N.$$

3. Написать функцию для вычисления суммы цифр целого числа, состоящего и произвольного количества цифр, без использования цикла.
4. Написать рекурсивную функцию для для подсчета числа сочетаний из N по K (C_N^K). Для решения задачи использовать тот факт, что $C_N^0 = C_N^N = 1$, а $C_N^K = C_{N-1}^K + C_{N-1}^{K-1}$.
5. Написать функции для решения некоторого нелинейного уравнения методами:
    а. половинного деления;
    б. Ньютона;
    в. хорд.
6. Написать функции для вычисления определенного интеграла на заданном отрезке от некоторой функции по квадратурным формулам:
    а. левых прямоугольников;
    б. правых прямоугольников;
    в. средних прямоугольников;
    г. трапеции;

д. Симпсона.

7. Написать шаблон функции для формирования двоичного кода некоторого целого числа в памяти компьютера. Функцией должны поддерживаться любые целые типы (char, bool, short, int, unsigned int, long long).

Тема 4. Работа с памятью. Организация хранения наборов данных в программах. Передача параметров в функции.

А. Теоретическая справка

Указатели

Указатель — это переменная, которая хранит не само значение определенного типа, а адрес памяти, где находится это значение.

Для объявления переменной-указателя перед ее именем ставится знак «*»:

```
1  int *p;//указатель на значения целого типа
2  float *x;//указатель на значения вещественного типа
3  void *t;//нетипизированный указатель
```

Указателю можно присвоить адрес уже существующей переменной или вновь выделенного под значение соответствующего типа блока памяти.

В следующем примере приводится вариант установки указателя на существующую переменную (адрес существующей переменной можно получить с помощью оператора «**&**»).

```
1  int a = 10;//Обычная целочисленная переменная
2  int* pa = &a;//Указатель получает адрес переменной a
```

Рассмотрим состояние памяти при выполнении указанных выше строк кода.

Строка 1. Создание целочисленной переменной.

	Состояние памяти								
Адрес	AF00	AF04	AF08	AF0C	AF10	AF14	AF18	AF1C	AF20
Значение	10								
Переменная	a								

Строка 2. Установка указателя на существующую переменную.

	Состояние памяти								
Адрес	AF00	AF04	AF08	AF0C	AF10	AF14	AF18	AF1C	AF20
Значение	10			AF00					
Переменная	a			*pa					

В случае вывода на экран значения указателя, будет отображен хранимый в этой переменной адрес. Если требуется получить значение, лежащее по этому адресу, необходимо применить операцию разыменования указателя («*****»).

```
1  cout << pa;//Будет выведен хранимый в переменной-указателе адрес
2  cout << *pa;//Будет выведено значение, лежащее по адресу, хранимому в указателе
```

Для того, чтобы выделить новую область памяти, не сопоставленную ни с одной из ранее объявленных переменных, следует воспользоваться оператором **new**. Он позволяет зарезервировать объем памяти, необходимый для хранения значения определенного типа и возвращает адрес выделенного блока. Занятую таким способом память необходимо освобождать после использования. Для этого существует оператор **delete**. Если программист забывает удалять неиспользуемые блоки памяти, в программе происходят так называемые утечки памяти. При продолжительной работе программы такая ситуация может со временем привести к нехватке памяти и снижению скорости работы всей системы в целом, либо аварийному завершению работы программы.

```
1  int* pn; //Объявление указателя
2  pn = new int; //Выделение памяти
3  *pn = 11; //Помещение значения в выделенную область памяти
4  cout << pn << ":" << *pn;   //Вывод на экран адреса и значения,
5                              //расположенного по соответствующему адресу
6  delete pn; //Освобождение памяти
```

При выполнении приведенного выше фрагмента кода состояние памяти будет изменяться следующим образом.

Строка 1. Объявляется указатель.

	Состояние памяти								
Адрес	AF00	AF04	AF08	…	FF00	FF04	FF08	FF0C	FF10
Значение									
Переменная	*pn								

Строка 2. Резервируется блок памяти по некоторому адресу (условно, FF00).

	Состояние памяти								
Адрес	AF00	AF04	AF08	…	FF00	FF04	FF08	FF0C	FF10
Значение	FF00				-9876[1]				
Переменная	*pn								

Строка 3. В ячейку по адресу FF00 помещается значение.

	Состояние памяти								
Адрес	AF00	AF04	AF08	…	FF00	FF04	FF08	FF0C	FF10
Значение	FF00				11				
Переменная	*pn								

ОБРАТИТЕ ВНИМАНИЕ! К ЯЧЕЙКЕ С АДРЕСОМ FF00 НЕВОЗМОЖНО ОБРАТИТЬСЯ ИНЫМ СПОСОБОМ, КРОМЕ КАК ЧЕРЕЗ УКАЗАТЕЛЬ **pn**, ПОСКОЛЬКУ ОТДЕЛЬНОГО СОПОСТАВЛЕННОГО С НЕЙ ИМЕНИ НЕ СУЩЕСТВУЕТ.

Строка 4. На экране будет отображен адрес (в нашем условном примере — FF00) и значение, лежащее по адресу (11).

Строка 6. Освобождение памяти по адресу FF00.

	Состояние памяти								
Адрес	AF00	AF04	AF08	…	FF00	FF04	FF08	FF0C	FF10
Значение	FFFF[2]								
Переменная	*pn								

В языке Си над указателями можно выполнять ряд арифметических операций. Адресная арифметика — это способ вычисления адреса какого-либо объекта при помощи арифметических операций над указателями, а также использование указателей в операциях сравнения.

Значение указателя можно увеличивать или уменьшать на некоторую целую величину. При этом, увеличение указателя на единицу будет соответствовать изменению хранимого в нем адреса на количество ячеек памяти, кратное размеру типа данных указателя.

Рассмотрим следующий пример.

[1] При выделении блока памяти в нем по умолчанию может храниться некоторое произвольное значение — «мусор».

[2] После очистки памяти указатель получает значение некоторого несуществующего адреса. Такой указатель называется «плохим» («bad pointer»).

Был выполнен фрагмент кода программы.

```
1  int a = 10;
2  int *pint = &a;
```

После этого состояние памяти приняло следующий вид.

	Состояние памяти										
Адрес	AF00	AF04	…	FF00	FF01	FF02	FF03	FF04	FF05	FF06	FF07
Значение	FF00			10							
Переменная	*pint			a							

Обратите внимание, что переменная a занимает в памяти 4 байта, поскольку именно такой объем отводится для целочисленных переменных. Указатель **`pint`** хранит адрес первого байта целого числа a.

Если выполнить увеличение указателя на единицу:

```
3  pint++;
```

то значение адреса, содержащегося в указателе **`pint`** станет равным не FF01, как можно было бы предположить, а FF04 — адреса памяти, где может находиться следующее целое число. Аналогичный ээфект происходит при уменьшении значения указателя.

Обратите внимание! Адресная арифметика не может быть применена к нетипизированным указателям (**`void *`**), поскольку заранее неизвестно, на какое количество ячеек следует изменять хранимые в них адреса.

Обратите внимание! Целочисленные переменные и указатели не являются совместимыми типами, несмотря на то, что последние содержат адреса, являющиеся, по сути, целочисленными значениями. Единственным исключением из этого правила является константа ноль: ее можно присвоить указателю, и указатель можно сравнить с нулевой константой. Однако, чтобы показать, что ноль — это специальное значение для указателя, вместо цифры ноль, как правило, записывают константу NULL.

Ссылки

Ссылка в C++ позволяет создать синоним (второе имя) для существующей переменной. При объявлении ссылки перед именем переменной ставится знак амперсанда (&). Объявляя ссылку, необходимо сразу же указать переменную, для которой эта ссылка создается, как показано ниже:

```
1  int a = 72; //Целочисленная переменная
2  int& na = a;//Ссылка на целочисленную переменную
```

ОБРАТИТЕ ВНИМАНИЕ! В ОТЛИЧИЕ ОТ УКАЗАТЕЛЯ, ССЫЛКА НЕ МОЖЕТ БЫТЬ ПУСТОЙ (НЕ СОДЕРЖАТЬ ЗНАЧЕНИЯ).

При выполнении вышеприведенных строк кода состояние памяти будет следующим.

Строка 1.

	Состояние памяти								
Адрес	AF00	AF04	AF08	…	FF00	FF04	FF08	FF0C	FF10
Значение	72								
Переменная	a								

Строка 2.

	Состояние памяти								
Адрес	AF00	AF04	AF08	…	FF00	FF04	FF08	FF0C	FF10
Значение	72								
Переменная	a								
Переменная	na								

ОБРАТИТЕ ВНИМАНИЕ! В ЯЗЫКЕ C/C++ НЕКОТОРЫЕ ЗНАКИ МОГУТ ОБОЗНАЧАТЬ РАЗНЫЕ ОПЕРАЦИИ В ЗАВИСИМОСТИ ОТ КОНТЕКСТА.

Рассмотрим возможные варианты применения одного и того же символа «&»:

```
1  int a = 0, b=1;
2  int &na = a; //Здесь «&» — это оператор создания ссылки
3  int* pa = &a; //Здесь «&» обозначает оператор получения адреса переменной
4  a=a & b; //Здесь — это операция побитовой конъюнкции для целых чисел
5  a=(a>0 & b>0)?a*b:a+b; //Здесь «&» — это логическое «И» в условной операции
```

Передача параметров в функции по указателям и ссылкам

Указатели и ссылки могут использоваться для организации передачи параметров в функцию. Рассмотрим это на следующем примере.

Предположим, программисту требуется написать функцию, которая должна поменять местами значения двух переменных. С этой целью был написан код:

```
1  #include <iostream>
2  using namespace std;
3
4  void swp(int a, int b){
5       int t = a;
```

```
6       a = b;
7       b = t;
8   }
9
10  void main(){
11      setlocale(LC_ALL, ".1251");
12      int a = 3, b = 7;
13      cout << "a=" << a << "; b=" << b << endl;
14      swp(a, b);
15      cout << "a=" << a << "; b=" << b << endl;
16      system("pause");
17  }
```

Вопреки ожиданиям программиста такая программа не выполнит требуемой операции. На экран будет выведено:

```
a = 3; b = 7;
a = 3; b = 7;
```

Рассмотрим, что будет происходить в памяти компьютера при ее выполнении.

Строка 12. Объявление переменных и их инициализация.

	Состояние памяти								
Адрес	AF00	AF04	AF08	AF0C	AF10	AF14	AF18	AF1C	AF20
Значение	3	7							
Переменная	a	b							
Область	main	main							

Строка 13. Вывод исходных значений переменных (a = 3; b = 7).

Строка 14. Вызов функции **swp**. Создание ее локальных переменных **a** и **b**. Копирование в формальные параметры значений фактических параметров.

	Состояние памяти								
Адрес	AF00	AF04	AF08	AF0C	AF10	AF14	AF18	AF1C	AF20
Значение	3	7			3	7			
Переменная	a	b			a	b			
Область	main	main			swp	swp			

Строка 5. Создание локальной переменной **t** и присвоение ей значения переменной **a**.

	Состояние памяти								
Адрес	AF00	AF04	AF08	AF0C	AF10	AF14	AF18	AF1C	AF20
Значение	3	7			3	7	3		
Переменная	a	b			a	b	t		
Область	main	main			swp	swp	swp		

Строка 6. Изменение значения переменной **a**.

	Состояние памяти								
Адрес	AF00	AF04	AF08	AF0C	AF10	AF14	AF18	AF1C	AF20
Значение	3	7			7	7	3		
Переменная	a	b			a	b	t		
Область	main	main			swp	swp	swp		

Строка 7. Изменение значения переменной **b**.

	Состояние памяти								
Адрес	AF00	AF04	AF08	AF0C	AF10	AF14	AF18	AF1C	AF20
Значение	3	7			7	3	3		
Переменная	a	b			a	b	t		
Область	main	main			swp	swp	swp		

Как видим, значения локальных переменных **a** и **b** функции **swp** действительно поменялись местами. Однако при выходе из функции, значения всех локальных переменных уничтожаются. При возврате к 15 строке остаются доступными только переменные функции **main**.

Строка 15. Вывод на экран значений переменных **a** и **b** функции **main**.

	Состояние памяти								
Адрес	AF00	AF04	AF08	AF0C	AF10	AF14	AF18	AF1C	AF20
Значение	3	7							
Переменная	a	b							
Область	main	main							

Они остались неизменными, и мы видим:

```
a = 3; b = 7
```

Эту проблему можно решить, если объявить параметры функции **swp** указателями. Соответственно, такой способ обмена данными называется передачей параметров по указателю.

```
1   #include <iostream>
2   using namespace std;
3
4   void swp(int* a, int* b){//Объявляем параметры-указатели
5         int t = *a;//В переменную копируем значение, получаемое по указателю.
6         *a = *b;//Необходимо поменять местами не указатели, а их значения,
7         *b = t;  //поэтому необходимо произвести операции разыменования
8   }
9
10  void main(){
11        setlocale(LC_ALL, ".1251");
12        int a = 3, b = 7;
13        cout << "a=" << a << "; b=" << b << endl;
14        swp(&a, &b); //Поскольку параметры функции swp теперь указатели,
15        //туда нужно передать не значения переменных, а их адреса.
```

```
16      cout << "a=" << a << "; b=" << b << endl;
17      system("pause");
18  }
```

Опять же проследим за изменением состояния памяти при работе программы.

Строка 12. Объявление переменных и их инициализация.

	Состояние памяти								
Адрес	AF00	AF04	AF08	AF0C	AF10	AF14	AF18	AF1C	AF20
Значение	3	7							
Переменная	a	b							
Область	main	main							

Строка 13. Вывод исходных значений переменных

```
a = 3; b = 7
```

Строка 14. Вызов функции **swp**. Создание ее локальных переменных-указателей **a** и **b**. Копирование значений фактических параметров в формальные. Только теперь в параметрах хранятся не целые числа 3 и 7, а адреса памяти, в которых они расположены.

	Состояние памяти								
Адрес	AF00	AF04	AF08	AF0C	AF10	AF14	AF18	AF1C	AF20
Значение	3	7			AF00	AF04			
Переменная	a	b			*a	*b			
Область	main	main			swp	swp			

Строка 5. Создание локальной переменной **t** и присвоение ей значения, расположенного по адресу AF00, записанному в указателе **a**.

	Состояние памяти								
Адрес	AF00	AF04	AF08	AF0C	AF10	AF14	AF18	AF1C	AF20
Значение	3	7			AF00	AF04	3		
Переменная	a	b			*a	*b	t		
Область	main	main			swp	swp	swp		

Строка 6. Изменение значения по адресу из указателя **a**.

	Состояние памяти								
Адрес	AF00	AF04	AF08	AF0C	AF10	AF14	AF18	AF1C	AF20
Значение	**7**	7			AF00	AF04	3		
Переменная	a	b			*a	*b	t		
Область	main	main			swp	swp	swp		

Строка 7. Изменение значения по адресу из указателя **b**.

	Состояние памяти								
Адрес	AF00	AF04	AF08	AF0C	AF10	AF14	AF18	AF1C	AF20
Значение	**7**	**3**			AF00	AF04	3		
Переменная	a	b			*a	*b	t		
Область	main	main			swp	swp	swp		

Заметим, что теперь поменялись значения исходных переменных **a** и **b** Далее, как и в прошлом примере, значения всех локальных переменных уничтожаются.

Строка 15. Вывод на экран значений переменных **a** и **b** функции **main**.

	Состояние памяти								
Адрес	AF00	AF04	AF08	AF0C	AF10	AF14	AF18	AF1C	AF20
Значение	7	3							
Переменная	a	b							
Область	main	main							

Теперь они изменились, и мы видим:

```
a = 7; b = 3
```

Программа работает!

Использование указателей позволило нам решить проблему с сохранением изменений параметров, произошедших внутри функции, в вызывающей подпрограмме. Однако при этом несколько усложнился синтаксис и ухудшилось восприятие кода (появилось много лишних символов «*» и «&».

Исправить этот недостаток можно благодаря использованию ссылок.

Код программы тогда будет записан следующим образом.

```
1   #include <iostream>
2   using namespace std;
3
4   void swp(int& a, int& b){//Объявляем параметры-ссылки
5         int t = a;//Ссылка это синоним обычной переменной,
6         a = b;//поэтому с ней можно работать как с обычной переменной,
7         b = t;  // не указателем, без использования лишних значков.
8   }
9
10  void main(){
11        setlocale(LC_ALL, ".1251");
12        int a = 3, b = 7;
13        cout << "a=" << a << "; b=" << b << endl;
14        swp(a, b);
15        cout << "a=" << a << "; b=" << b << endl;
16        system("pause");
17  }
```

Рассмотрим, как и прежде, изменения, происходящие в памяти начиная с четырнадцатой строки (до этого отличий от предыдущих примеров нет).

Строка 14. Вызывается функция **swp**. Ее формальные параметры **a** и **b** становятся ссылками на переменные **a** и **b** функции **main**.

	Состояние памяти								
Адрес	AF00	AF04	AF08	AF0C	AF10	AF14	AF18	AF1C	AF20
Значение	3	7							
Переменная	a	b							
Область	main	main							
Переменная	a	b							
Область	swp	swp							

Строка 5. Создание локальной переменной **t** и присвоение ей значения, переменной a.

	Состояние памяти								
Адрес	AF00	AF04	AF08	AF0C	AF10	AF14	AF18	AF1C	AF20
Значение	3	7			3				
Переменная	a	b			t				
Область	main	main			swp				
Переменная	a	b							
Область	swp	swp							

Строка 6. Изменение значения переменной **a** функции **swp** и одновременно — значения ее синонима — переменной **a** функции **main**.

	Состояние памяти								
Адрес	AF00	AF04	AF08	AF0C	AF10	AF14	AF18	AF1C	AF20
Значение	7	7			3				
Переменная	a	b			t				
Область	main	main			swp				
Переменная	a	b							
Область	swp	swp							

Строка 7. Изменение значения переменной **b**.

	Состояние памяти								
Адрес	AF00	AF04	AF08	AF0C	AF10	AF14	AF18	AF1C	AF20
Значение	7	3			3				
Переменная	a	b			t				
Область	main	main			swp				
Переменная	a	b							
Область	swp	swp							

Как и положено, значения всех локальных переменных уничтожаются при выходе из функции.

Строка 15. Вывод на экран значений переменных **a** и **b** функции **main**.

	Состояние памяти								
Адрес	AF00	AF04	AF08	AF0C	AF10	AF14	AF18	AF1C	AF20
Значение	7	3							
Переменная	a	b							
Область	main	main							

На экране видим:

```
a = 7; b = 3
```

Создание статических массивов

Массив — это набор однотипных компонентов (элементов), расположенных в памяти непосредственно друг за другом, доступ к которым осуществляется по общему имени и одному или нескольким индексам. По сути, массив представляет собой переменную, которая позволяет сохранять вместе несколько значений одного и того же типа.

Размерностью массива называется количество индексов, которое необходимо использовать для получения доступа к конкретному элементу.

Массивы в языке Си и Си++ могут быть статическими (они хранятся в области локальных или глобальных переменных, и их размер должен быть известен до компиляции) и динамическими (они хранятся в динамической памяти, так называемой «куче», и их размер можно задать во время выполнения программы, непосредственно в момент создания).

Для описания статического массива в программе необходимо написать следующий (или подобный) код:

```
1   #define n 5
2   int mas[n];
```

В приведенном примере создается константа n равная 5, а затем создается массив, который сможет содержать n (то есть, в данном случае, 5) целых чисел.

В дальнейшем для работы с массивом необходимо будет обращаться к каждому его элементу, указывая общее имя массива и номер конкретного элемента в квадратных скобках. Например,

```
1   mas[0] = 14;
2   mas[2] = 21;
```

В языке C/C++ нумерация массивов всегда начинается с 0. Таким образом, индекс первого элемента будет 0, а последнего — n-1, где n — размер массива.

При создании массива у программиста есть возможность сразу определить значения всех его элементов, перечислив их в фигурных скобках после знака равенства:

```
1   char ops[] = {'+', '-', '*', '/'};
```

Данный массив содержит список из четырех символов знаков операций.

Обратите внимание! При задании массива с указанием всех его значений, количество элементов в фигурных скобках можно не указывать.

Следующая программа демонстрирует работу с массивом. В ней создается новый массив, он заполняется значениями, введенными с клавиатуры, а затем вычисляется и выводится на экран сумма элементов.

```
1   #include <iostream>
2   using namespace std;
3
4   //Константа N глобальной области видимости
5   //и будет доступна из любой функции программы
6   const int N = 5;
7
8   //Вычисляет сумму элементов массива
9   float sum(int mas[]){
10        float s = 0;
11        //Находим сумму элементов
12        for (int i = 0; i < N; i++){
13              s += mas[i];
14        }
15        return s;
16  }
17
18  void main(){
19        setlocale(LC_ALL, ".1251");
20        int a[N];
21        cout << "Введите элементы массива:\n";
22        for (int i = 0; i < N; i++){
23              cout << "a[" << i << "]=";
24              //Вводим элементы массива:
25              cin >> a[i];
26        }
27        float sa = sum(a);
28        cout << "Сумма элементов массива=" << sa << endl;
29        system("pause");
30  }
```

Создание динамических массивов, их связь с указателями

Для создания динамических массивов в языке C++ используется оператор **new[]**. Он позволяет указать тип элементов массива и количество его элементов. Оператор возвращает указатель на адрес первой ячейки памяти, которая была выделена под массив.

ОБРАТИТЕ ВНИМАНИЕ! ВСЕ ЭЛЕМЕНТЫ МАССИВА РАСПОЛАГАЮТСЯ В ПАМЯТИ ПОДРЯД. ЕСЛИ ОПЕРАЦИОННАЯ СИСТЕМА НЕ МОЖЕТ ОБНАРУЖИТЬ НЕПРЕРЫВНУЮ ОБЛАСТЬ ПАМЯТИ НЕОБХОДИМОГО РАЗМЕРА, ТО МАССИВ НЕ БУДЕТ СОЗДАН, ДАЖЕ ЕСЛИ ОБЩЕГО КОЛИЧЕСТВА СВОБОДНОЙ ПАМЯТИ ДОСТАТОЧНО.

Пример описания вещественнозначного динамического массива:

```
1  int n;
2  cout << "Введите количество элементов массива: ";
3  cin >> n;
4  float* mas = new float[n];
5  //…
```

Обратите внимание, что число элементов динамического массива в примере вводится с клавиатуры.

Если использование массива в программе далее не планируется, выделенную под него память необходимо очистить с помощью оператора **delete[]**.

```
6  delete [] mas;
```

Работа с динамическими массивами может осуществляться точно так же, как и с массивами статическими, то есть для получения доступа к их значениям нужно указать имя переменной и далее, в квадратных скобках, индекс конкретного элемента.

В языке Си массивы настолько тесно связаны с указателями, что доступ к элементам массива может быть записан даже без использования квадратных скобок (с применением адресной арифметики).

Так, получить доступ к первому элементу массива можно при помощи операции разыменования указателя:

```
1  int n = 3;
2  int* mas = new int[n];
3  *mas = 19; //Изменяем первый элемент массива.
```

Для доступа к последующим элементам можно изменять значения указателя:

```
4   int* m = mas;//Сохраняем указатель на начало массива
5   mas++;//Теперь указатель mas ссылается не на нулевой, а на первый элемент массива!
6   *mas = 10;
7   mas+=1; //Перемещаем указатель на следующий элемент
8   *mas = *(mas-2)+*(mas-1); //Последний элемент будет равен сумме 1-го и 2-го
9   cout << *mas << endl; //Будет выведено число 29.
```

После работы с массивом необходимо очистить память. Однако, указатель **mas** уже «испорчен», поскольку был изменен и теперь ссылается не на начало массива. Поэтому для освобождения памяти нужно использовать сохраненную копию указателя — **m**.

```
10  delete [] m;
```

Обратите внимание! Из последнего примера видно, что обращение к элементу массива через квадратные скобки: **mas[i]** фактически эквивалентно разыменованию измененного указателя: ***(mas+i)**. Поскольку от перемены мест слагаемых сумма, как известно, не меняется, допустима и такая запись: ***(i+mas)**, а значит, обращение к элементу массива **mas** с номером **i** может быть выполнено и таким образом: **i[mas]**.

Очень часто в программах при передаче динамического массива в функцию возникает вопрос, как в этой функции определить длину массива. Обычно в литературе предлагается передавать в функцию размер массива в виде отдельной переменной. Однако, есть и другая возможность. Размер выделенного блока динамической памяти в байтах можно узнать с помощью функции **_msize()**, которая принимает в качестве параметра указатель на выделенный блок динамической памяти. Таким образом, чтобы узнать число элементов в массиве можно поделить число байт, отведенных для всего массива, на число байт, отводимых для одного его элемента. Последнюю величину можно узнать при помощи оператора **sizeof**.

```
1   //Функция поиска минимального элемента массива
2   float min(float* mas){
3         float min = *mas;//Сначала считаем первый элемент минимальным
4         //Находим количество элементов в массиве
5         int N = _msize(mas)/sizeof(mas);
6         //Сравниваем все элементы массива с наименьшим:
7         for (int i = 1; i < N; i++){
```

```
8                   //Если найден элемент меньше минимального,
9                   //то обновляем минимальный элемент.
10                  if (min>mas[i]) min = mas[i];
11              }
12              return min;//Возвращаем найденное значение
13      }
```

Обратите внимание! Функция **`_msize`** будет корректно работать только для указателей, содержащих адреса из динамической области памяти (кучи).

Функции с переменным числом параметров

Использование указателей позволяет программисту создавать функции с переменным числом параметров. Примерами таких функций являются известные вам **`printf_s`** и **`scanf_s`**, осуществляющие консольный ввод/вывод данных произвольных типов в заранее неопределенном количестве.

Описание функции с переменным числом параметров должно содержать один или более обязательных аргументов, после которых ставится знак многоточия (**…**).

По понятным причинам, к тем параметрам, которые не являются обязательными, программист не имеет возможности обратиться по имени. Поэтому для обеспечения доступа к ним необходимо выполнить ряд операций.

1. Установить указатель на последний обязательный параметр. При этом тип указателя должен соответствовать типу самого аргумента.
2. Учитывая тот факт, что все параметры, передаваемые в функцию, расположены в памяти подряд, увеличить значение указателя на единицу (по правилам адресной арифметики, будет получен адрес значения, следующего за значением последнего обязательного параметра).
3. Повторяя эту операцию, получить значения всех последующих параметров, при условии, что все параметры будут иметь одинаковый тип данных.

Обратите внимание! Для того, чтобы программа могла работать корректно, она должна располагать информацией о типах и количестве переданных параметров. Если они отличаются от типа обязательного параметра, то при увеличении значения указателя, необходимо будет учесть разницу в количестве памяти, занимаемой различными типами.

Более подробный пример использования функций с переменным числом аргументов можно посмотреть ниже, в примере № 2 из части Б.

Указатели на функции

Функция, как и любой другой объект в программе, занимает определенное место в памяти. Поэтому на функцию можно создать указатель. Указатели на функции — это переменные, которые содержат в качестве значения адрес функции, а точнее — адрес точки входа в функцию.

Пусть имеются две функции Plus и Minus:

```
1  double Plus(double a, double b){ return a+b;}
2  double Minus(double a, double b){return a-b;}
```

Тогда для задания указателя на функцию можно использовать следующий синтаксис:

```
3  double (*pt2Function)(double, double) = NULL;
```

В данном случае определяется указатель с именем pt2Function, а сами функции, на которые он может ссылаться должны принимать два аргумента типа double и возвращать значение такого же типа. При этом вновь определенный указатель в нашем примере не получает никакого конкретного адреса, а инициализируется значением NULL.

Для присвоения указателю адреса конкретной функции можно написать:

```
4  pt2Function = &Plus;
```

Вызов функции по указателю при этом осуществляется следующим образом:

```
5  double x = (*pt2Function)(10., 20.);
6  cout << x << endl; //30
```

Также можно использовать сокращенные варианты записей этих действий, но они могут работать не на всех версиях компиляторов.

```
7  pt2Function = Minus;
8  double y = pt2Function(10., 20.);
9  cout << y << endl; //-10
```

Указатель на функцию также можно использовать для передачи соответствующего параметра в другую функцию:

```
1   double Plus(double a, double b){ return a+b;}
2   double Minus(double a, double b){return a-b;}
3   double doOperation(double a, double b,
4                      double (*pt2Fn)(double, double)){
5        return pt2Fn(a, b);
6   }
```

Сама функция doOperation может быть использована следующим образом:

```
7   //…
8   cout << doOperation(1, 3, &Plus) << endl; //4
```

В случаях, когда требуется, чтобы функция возвращала указатель на функцию, лучше всего воспользоваться инструкцией определения нового типа **`typedef`**:

```
1   double Plus(double a, double b){ return a+b;}
2   double Minus(double a, double b){return a-b;}
3   //Определение типа-указателя на функцию
4   typedef double (*pt2Func)(double, double);
5   //Функция, возвращающая указатель соответствующего типа:
6   pt2Func Select(char op){
7        switch (op){
8        case '+': return &Plus;
9        case '-': return &Minus;
10       }
11       return NULL;
12  }
13  //…
14  //Пример использования функции:
15  cout << doOperation(44, 56, Select('+')) << endl; //100
```

Б. Примеры программ

1. Дано время (количество часов (h), минут (m) и секунд (s)). Требуется написать функцию для добавления к этому времени t секунд.

```
1   #include <iostream>
2   using namespace std;
3
4   //Добавление t секунд ко времени,
5   //заданному в часах (h), минутах (m) и секундах (s)
6   void AddTime(unsigned *h, unsigned *m,
7                unsigned *s, unsigned t){
8        //Для сохранения изменений в параметрах, передаем их через указатели
9        unsigned extraM;
```

```
10        //Получаем дополнительное количество минут, образованных секундами
11        extraM = (*s + t) / 60;
12        //Вычисляем оставшееся количество секунд
13        *s = (*s + t)%60;
14        unsigned extraH;
15        //Получаем дополнительное количество часов, образованных минутами
16        extraH = (*m + extraM) / 60;
17        //Вычисляем оставшееся количество минут
18        *m = (*m + extraM) % 60;
19        //Вычисляем полученное число часов
20        *h = *h + extraH;
21  }
22
23  void main(){
24        setlocale(LC_ALL, ".1251");
25        //Начальное время (часы, минуты, секунды)
26        unsigned h = 22, m = 54, s = 47;
27        cout << h << ":" << m << ":" << s; //22:54:47
28        cout << " + 10000 сек. = ";
29        //Добавляем 10000 секунд
30        AddTime(&h, &m, &s, 10000);
31        //Выводим полученное время, сохраненное в тех же переменных
32        cout << h << ":" << m << ":" << s << endl; //25:41:27
33        system("pause");
34  }
```

2. Написать функцию, вычисляющую наибольшее значение среди произвольного количества параметров вещественного типа, значение последнего из которых (и только его) устанавливается равным 0.

```
1   #include <iostream>
2   using namespace std;
3
4   double max(double fst, ...){
5         double* ptr = &fst;
6         double m = fst;
7         //Запускаем цикл, работающий, пока значение, находящееся
8         //по адресу, увеличенному на 1, не равно 0.
9         while (*(++ptr)){
10              if (m<*ptr) m = *ptr;
11        }
12        return m;
13  }
14
15  void main(){
16        setlocale(LC_ALL, ".1251");
17        double s = max(3., 43.5, 40.1, 31.34, 0.98,
18                              4.32, 5.212, 444.23, -948.43, 0.);
19        cout << "Наибольшее из чисел:\n"
20                  "3., 43.5, 40.1, 31.34, 0.98, "
21                  "4.32, 5.212, 444.23, -948.43 = " << s << endl;
22        system("pause");
23  }
```

Результат работы программы:

```
Наибольшее из чисел:
3., 43.5, 40.1, 31.34, 0.98, 4.32, 5.212, 444.23, -948.43 = 444.23
```

3. Дан массив, содержащий 10 элементов (элементы массива вводятся с клавиатуры). Найти их произведение.

```
1   #include <iostream>
2   using namespace std;
3
4   void main(){
5         setlocale(LC_ALL, ".1251");
6         const int N = 10;//Число элементов массива
7         int a[N];//Объявление статического массива
8         //Ввод элементов массива:
9         for (int i = 0; i < N; i++){
10              cout << "a[" << i << "]=";
11              cin >> a[i];
12        }
13        //Нахождение произведения элементов:
14        int p = 1;//Начальное значение произведения
15        for (int i = 0; i < N; i++){
16              //Перебирая элементы по очереди, находим произведения
17              p *= a[i];
18        }
19        cout << "Произведение элементов = " << p << endl;
20        system("pause");
21  }
```

Результат работы программы:

```
a[0]=3
a[1]=1
a[2]=2
a[3]=4
a[4]=5
a[5]=6
a[6]=7
a[7]=8
a[8]=9
a[9]=10
Произведение элементов = 3628800
```

4. Дан массив, содержащий N элементов (число N и элементы массива вводятся с клавиатуры). Написать функцию для нахождения среднего арифметического значения элементов массива.

```
1   #include <iostream>
2   using namespace std;
3
4   double Avg(int *a, int n){//Передаем указатель на массив и его размер
5         double avg = 0.;
```

```
6       for (int i = 0; i < n; i++){
7            avg += a[i];//Вычисляем сумму элементов
8       }
9       return avg / n;//Возвращаем среднее арифметическое
10  }
11
12  void main(){
13       setlocale(LC_ALL, ".1251");
14       cout << "Введите количество элементов массива: ";
15       int n;
16       cin >> n;
17       //Создаем динамический массив:
18       int* a = new int[n];
19       for (int i = 0; i < n; i++){
20            cout << "a[" << i << "]=";
21            cin >> a[i];
22       }
23       double avg = Avg(a, n);
24       cout << "Ср. арифм. элементов массива = " << avg << endl;
25       //Освобождаем память, занятую массивом
26       delete[] a;
27       system("pause");
28  }
```

Результат работы программы:

```
Введите количество элементов массива: 5
a[0]=3
a[1]=7
a[2]=5
a[3]=4
a[4]=2
Ср. арифм. элементов массива = 4.2
```

5. Написать программу, содержащую следующие функции для работы с веществennозначным массивом:

а) создания массива заданного размера;

б) заполнения массива значениями с клавиатуры;

в) отображения элементов массива на клавиатуре;

г) удаления массива;

д) сортировки элементов массива.

Функции для работы с массивом разместить в отдельном файле.

```
1   //massiv.h
2   void Create(double *&mas, int n);
3   void Fill(double *mas);
4   void Show(double* mas);
5   void Delete(double *&mas);
6   void Sort(double *mas);
7   void Swap(double &a, double &b);
```

```
1   //massiv.cpp
2   #include <malloc.h>
3   #include <iostream>
4   using namespace std;
5   #include "massiv.h"
6
7   //Создание массива
8   void Create(double *&mas, int n){
9         //При создании массива меняется значение самого указателя,
10        //поэтому параметр-указатель mas должен быть передан по ссылке.
11        mas = new double[n];
12  }
13
14  //Заполнение массива данными с клавиатуры
15  void Fill(double *mas){
16        if (!mas) return;
17        int n = _msize(mas)/sizeof(*mas);//Получаем размер массива:
18        cout << "Введите " << n << " элементов:\n";
19        //Цикл для ввода элементов:
20        for (int i = 0; i < n; i++)
21        {
22              cout << "Элемент №" << i << ": ";
23              cin >> mas[i];
24        }
25  }
26  //Вывод содержимого массива на экран
27  void Show(double *mas){
28        int n = _msize(mas) / sizeof(*mas);//Получаем размер массива:
29        //Цикл для вывода элементов:
30        for (int i = 0; i < n; i++)
31        {
32              cout << mas[i] << ((i==n-1)?"":" ");
33        }
34        cout << endl;
35  }
36
37  //Удаление массива из памяти
38  void Delete(double *&mas){
39        if (mas) //Если массив создан
40              delete[] mas;
41        mas = NULL;
42  }
43
44  //Смена местами значений двух переменных
45  void Swap(double &a, double &b){
46        double t = a;
47        a = b;
48        b = t;
49  }
50
51  //Сортировка массива по возрастанию
52  void Sort(double *mas){
53        int n = _msize(mas) / sizeof(*mas);//Получаем размер массива:
54        //Практически любая сортировка требует применения вложенных циклов
55        for (int i = 0; i < n - 1; i++){
```

```
56              for (int j = i + 1; j < n; j++){
57                    //Обратите внимание, что i всегда меньше j!
58                    if (mas[i]>mas[j]){
59                          //Если элемент, расположенный слева, больше элемента,
60                          //находящегося правее, их нужно поменять местами.
61                          Swap(mas[i], mas[j]);
62                    }
63              }
64        }
65  }
```

```
1.  //main.cpp
2.  #include <iostream>
3.  using namespace std;
4.  #include "massiv.h"
5.
6.  void main(){
7.        setlocale(LC_ALL, ".1251");
8.        double *x = NULL;
9.        int n;
10.       cout << "Введите размер массива: ";
11.       cin >> n;
12.       Create(x, n);//Создаем массив из n элементов
13.       Fill(x);//Заполняем массив значениями с клавиатуры
14.       cout << "Исходный массив:        ";
15.       Show(x);//Отображаем содержимое массива на экране
16.       Sort(x);//Сортируем массив
17.       cout << "Отсортированный массив: ";
18.       Show(x);//Отображаем содержимое массива после сортировки
19.       Delete(x);//Удаляем массив
20.       system("pause");
21. }
```

Результат работы программы:

```
Введите размер массива: 5
Введите 5 элементов:
Элемент №0: 45
Элемент №1: -98
Элемент №2: 34
Элемент №3: -8
Элемент №4: 11
Исходный массив:        45 -98 34 -8 11
Отсортированный массив: -98 -8 11 34 45
```

В. ЗАДАНИЯ ДЛЯ САМОСТОЯТЕЛЬНОГО ВЫПОЛНЕНИЯ

1. Описать функцию TimeToHMS, определяющую по времени T (в секундах) содержащееся в нем количество часов (H), минут (M) и секунд (S).

Используя эту функцию, найти количество часов, минут и секунд для пяти данных отрезков времени.

2. Написать функцию, добавляющую слева к целому положительному числу K цифру D ($1 \le D \le 9$). С помощью этой функции последовательно добавить к некоторому числу три цифры.
3. Написать функцию, меняющую содержимое трех своих параметров таким образом, чтобы их значения оказались упорядоченными по возрастанию.
4. Описать функцию, зависящую от трех параметров, выполняющую правый циклический сдвиг: значение первого параметра переходит во второй, второго — в третий, третьего — в первый.
5. Дан массив размера N и целые числа K и L ($0 < K \le L < N$). Найти среднее арифметическое всех элементов массива, кроме элементов с номерами от K до L включительно.
6. Дан целочисленный массив размера N, все элементы которого упорядочены по возрастанию или убыванию. Найти количество различных элементов в данном массиве.
7. Дан массив размера N. Найти максимальный из его локальных минимумов (элемента, меньшего любого из своих соседей).
8. Дано множество A из N точек на плоскости и точка B (точки заданы своими координатами x, y). Найти точку из множества A, наиболее близкую к точке B. Расстояние R между точками с координатами (x_1, y_1) и (x_2, y_2) вычисляется по формуле:

$$R = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}.$$

9. Даны три целочисленных массива различных размеров, элементы которых упорядочены по возрастанию. Объединить эти массивы так, чтобы результирующий целочисленный массив остался упорядоченным по возрастанию.

10. Написать функции для сортировки массива методами пузырька, Шелла, выбора, пирамидальной сортировки, блочной сортировки, быстрой сортировки.
11. Написать программу, которая будет предлагать осуществлять сортировку массива различными методами (см. п. 10). Выбор метода сортировки организовать через указатель на функцию.

ТЕМА 5. РАБОТА С НУЛЬ-ТЕРМИНАЛЬНЫМИ СТРОКАМИ.

А. ТЕОРЕТИЧЕСКАЯ СПРАВКА

Понятие нуль-терминальных строк

В языке программирования Си нет отдельного типа для хранения строковых данных (соответствующий тип, а точнее класс, появляется только в языке C++). Вместо этого программист может создать массив, состоящий из элементов символьного типа. При этом для работы со строкой могут быть использованы все те же приемы, которые применяются при работе с остальными массивами.

Однако строки по сравнению с остальными массивами обладают расширенной функциональностью. Достигается это благодаря тому, что строки в Си являются нуль-терминальными (или нуль-терминированными). В них обязан содержаться специальный символ, обозначающий конец строки. В качестве такового используется специальный символ с кодом ноль (терминальный ноль). Без терминального нуля последовательность символов не сможет обрабатываться как строка, так как не будет известно, где она заканчивается.

Так, строки могут быть объявлены следующими способами:

```
1  const char *s0 = "Hi!"; //Константная строка
2  char *s1 = new char[6]; //Строка длиной до 5(!) символов:
3  //последняя ячейка резервируется для терминального нуля
4  s1[0]='H'; s1[1]='i'; s1[2] = '!';
```

Рассмотрим, как эти строки будут представлены в памяти.

Строка 1.

	Состояние памяти												
Адрес	AF00	AF01	AF02	AF03	AF04	AF05	…	FF01	FF02	FF03	FF04	FF05	FF06
Значение	AF02		'H'	'i'	'!'	'\0'							
Переменная	s0												

Строка 2.

	Состояние памяти												
Адрес	AF00	AF01	AF02	AF03	AF04	AF05	…	FF01	FF02	FF03	FF04	FF05	FF06
Значение	AF02	FF01	'H'	'i'	'!'	'\0'		'H'[1]	'H'	'H'	'H'	'э'	'э'
Переменная	s0	s1											

Строка 3. Помещаем посимвольно в строку **`s1`** те же данные, что были в строке **`s0`**.

	Состояние памяти												
Адрес	AF00	AF01	AF02	AF03	AF04	AF05	…	FF01	FF02	FF03	FF04	FF05	FF06
Значение	AF02	FF01	'H'	'i'	'!'	'\0'		'H'	'i'	'!'	'H'	'э'	'э'
Переменная	s0	s1											

Однако результат вывода на экран для этих двух строк будет разным:

```
5  cout << s0 << endl; //Hi!
6  cout << s1 << endl; //Hi!HHээ
```

Такое различие объясняется тем, что указатель **`s0`** ссылается на строку-константу, в которую уже был автоматически добавлен символ терминаль-

[1] Сразу после выделения памяти в ней может содержаться мусор — значения оставшиеся там от прошлых строк или иных данных.

ного нуля. Однако, содержимое строки **s1** формировалось нами посимвольно и терминальный ноль в ней отсутствует. Поэтому на экран был выведен мусор, который находился в этой строке после значащей части.

Для устранения этого недостатка необходимо добавить в строку **s1** терминальный ноль:

```
7    s1[3] = 0;
8    cout << s1 << endl; //Hi!
```

После выполнения действия, записанного в строке 6, мусор, находящийся в оставшейся части строки, игнорируется.

ОБРАТИТЕ ВНИМАНИЕ! ВЫВОД СОДЕРЖИМОГО СТРОКИ ОСУЩЕСТВЛЯЕТСЯ ПРОСТО ПО ИМЕНИ УКАЗАТЕЛЯ (**cout << s1;**) ДЛЯ УКАЗАТЕЛЯ ЛЮБОГО ДРУГОГО ТИПА ТАКАЯ ЗАПИСЬ ПРИВЕЛА БЫ К ПОЯВЛЕНИЮ НА ЭКРАНЕ АДРЕСА ПАМЯТИ.

При объявлении и присвоении значений строкам, студенты часто совершают ошибку, приводящую к утечке памяти. Рассмотрим ее на следующем примере.

```
1    char* str = new char[256];
2    str = "Привет, МИР!";
3    //…
4    delete [] str;
```

Первая ошибка, которая может сразу не проявиться, заключается в том, что вместо того, чтобы записать содержимое «Привет, МИР!» в существующую строку **str** длиной до 256 символов, программист лишь меняет значение указателя **str**. То есть после выполнения второй строки указатель получает новый адрес (адрес, где хранится константа «Привет, МИР!»), теряя при этом указатель на область памяти в 256 символов, первоначально выделенную для строки. В дальнейшем, при попытке освободить блок памяти **str** оператором **delete[]**, возникает ошибка времени выполнения, поскольку данный блок не выделен динамически, а является константой.

Для того, чтобы изменить значение существующей строки, необходимо либо обращаться к ней посимвольно (что не всегда удобно), либо применять специальные функции для работы со строками, которые будут рассмотрены ниже.

Также необходимо сделать несколько замечаний относительно способа чтения строки с клавиатуры. Безусловно, с этой целью можно использовать стандартную операцию языка Си++.

```
1    char* str = new char[256];
2    cin >> str;
```

Либо, как вариант, использовать стиль языка Си:

```
2    scanf_s("%s", str, 256);
```

Однако, следует иметь в виду, что при этом в переменной окажется только часть строки, находящаяся до первого пробела, который воспринимается как символ-разделитель. В этом легко убедиться, если вывести строку на экран.

Если в строку **`str`** попытаться ввести фразу «Программирование — это гонка между компьютерщиками, которые создают программы, все лучше защищенные от дурака, и природой, которая создает все лучших дураков. Пока что природа выигрывает.», то после выполнения команды

```
3    cout << str;
```

на экране отобразится только слово «Программирование».

Чтобы прочитать все предложение, можно воспользоваться одним из следующих способов:

```
4    cin.getline(str, 256);
```

либо, если вы предпочитаете использовать стиль языка Си, а не Си++:

```
4    gets_s(str, 256);
```

Тогда текст будет считан до появления символа конца строки (**`'\n'`**).

ОБРАТИТЕ ВНИМАНИЕ! ЧИСЛО 256, ВЫСТУПАЮЩЕЕ ПАРАМЕТРОМ ФУНКЦИЙ ЧТЕНИЯ ДАННЫХ С КЛАВИАТУРЫ, ОБОЗНАЧАЕТ МАКСИМАЛЬНО ДОСТУПНЫЙ ОБЪЕМ, КОТОРЫЙ МОЖЕТ БЫТЬ ПРОЧИТАН. ЭТОТ ОБЪЕМ НЕ ДОЛЖЕН ПРЕВЫШАТЬ РАЗМЕР МАССИВА, СОЗДАННОГО ПРИ ПОМОЩИ ОПЕРАТОРА **`new[]`**.

Функции для работы со строками

Для работы с нуль-терминальными строками в языке Си используются специальные функции, названия, прототипы и описания которых приведены ниже.

Таблица 14.

Основные функции для работы со строками

№	Функция	Прототип и краткое описание
1.	**strcat_s**	**errno_t strcat_s(** **char *strDestination,** **size_t[1] numberOfElements,** **const char *strSource** **);** Приписывает строку strSource к строке strDestination. numberOfElements — размер массива strDestination — должен быть достаточным для хранения значения объединенных строк.
2.	**strchr**	**char *strchr(** **char * str,** **int c** **);** Ищет в строке str первое вхождение символа c и возвращает указатель на первое вхождение этого символа в строке str.
3.	**strcmp**	**int strcmp(** **const char *string1,** **const char *string2** **);** Сравнивает две строки друг с другом. Возвращает отрицательное значение, если string1 < string2; положительное значение, если string1 > string2; 0, если строки идентичны. Сравнение строк учитывает как их длину, так и содержание.
4.	**strcpy_s**	**errno_t strcat_s(** **char *strDestination,** **size_t numberOfElements,** **const char *strSource** **);** Копирует содержимое строки strSource в строку strDestination. numberOfElements — размер массива strDestination — должен быть достаточным для хранения строки strSource.
5.	**_strdup**	**char *_strdup(** **const char *strSource** **);** Создает новую строку, выделяя память необходимого размера и копирует в нее содержимое строки strSource. Возвращает указатель на новую строку или NULL, если память не может быть выделена.

[1] Тип size_t эквивалентен типу unsigned int.

6.	**strlen**	**size_t strlen(** **const char *str** **);** Получает количество символов, находящихся в строке str до терминального нуля.
7.	**strncmp**	**int strncmp(** **const char *string1,** **const char *string2,** **size_t count** **);** Сравнивает до count символов двух строк друг с другом. Возвращает отрицательное значение, если подстрока string1 < string2; положительное значение, если подстрока string1 > string2; 0, если подстроки длины count идентичны.
8.	**strnicmp**	**int strnicmp(** **const char *string1,** **const char *string2,** **size_t count** **);** Сравнивает до count символов двух строк друг с другом. Возвращает отрицательное значение, если подстрока string1 < string2; положительное значение, если подстрока string1 > string2; 0, если подстроки длины count идентичны. При сравнении регистр букв не учитывается.
9.	**strrchr**	**char *strrchr(** **char * str,** **int c** **);** Ищет в строке str последнее вхождение символа c и возвращает указатель на последнее вхождение этого символа в строке str.
10.	**strstr**	char *strstr(char *str, const char *strSearch); Возвращает указатель на первое вхождение подстроки strSearch в строке str или NULL, если подстрока в строке не найдена.
11.	**strtok_s**	**char *strtok_s(** **char *strToken,** **const char *strDelimit,** **char **context** **);** Разделяет строку strToken на части, используя в качестве разделителей символы из строки strDelimit. При первом вызове функции в первом параметре указывается делимая строка, в последующих — NULL; после каждого отделения указатель на оставшуюся часть строки сохраняется в context.

Преобразование типов

Зачастую в программах требуется осуществить преобразование данных из строкового типа в числовой и обратно. Для этих целей можно воспользоваться универсальными функциями форматированного ввода и вывода данных для строк: **sscanf_s** и **sprintf_s**.

Аналогичные функции (без приставки «s»), предназначенные для организации консольного ввода/вывода, были нами рассмотрены выше, на стр. 43. Отличие в использовании функций **sscanf_s** и **sprintf_s** заключается лишь в том, что первым параметром у них ставится строка из которой данные получаются или в которую они будут записываться.

Прототипы указанных функций приводятся ниже:

```
int sscanf_s(const char *buffer, const char *format [, arg ]...);
int sprintf_s(char *buffer, size_t sz, const char *format
                                                    [,arg]...);
```

Флаги, используемые в строке формата приведены в таблице 12 на странице 47.

Рассмотрим пример. Пусть дана строка, содержащая число, месяц и год в формате «дд.мм.гггг». Требуется получить в целочисленных переменных значения месяца и года. Для этого можно использовать следующий код:

```
1  char *date = "18.03.2008";
2  int mon, year;
3  sscanf_s(date, "%*d.%d.%d", &mon, &year);
```

Наоборот, если, например, требуется вывести время в формате «ч:мм:сс» по заданным числовым значениям, можно будет написать:

```
1  int h = 9;
2  int m = 20;
3  int s = 0;
4  char *tm = new char[9];
5  sprintf_s(tm, 9, "%d:%s%d:%s%d", h, (m>9)?"":"0", m,
6           (s>9)?"":"0", s);
```

ОБРАТИТЕ ВНИМАНИЕ! ВНУТРИ ФУНКЦИИ **sprintf_s** БЫЛА ПРИМЕНЕНА УСЛОВНАЯ ОПЕРАЦИЯ С ЦЕЛЬЮ ДОБАВЛЕНИЯ ВЕДУЩЕГО НУЛЯ ПЕРЕД МИНУТАМИ И СЕКУНДАМИ, ЕСЛИ ОНИ ПРЕДСТАВЛЯЮТСЯ ОДНОЗНАЧНЫМ ЧИСЛОМ.

Б. Примеры программ

1. Дана строка, содержащее некоторое слово или предложение. Требуется определить, является ли оно палиндромом (читается одинаково как справа налево, так и слева направо).

```
1   #include <iostream>
2   #include "Windows.h"
3   using namespace std;
4
5   void main(){
6         SetConsoleCP(1251);
7         SetConsoleOutputCP(1251);
8         int length;
9         do {//Будем проверять предложения, пока не будет введена пустая строка
10              cout << "Введите строку длиной до 255 символов: ";
11              //Создаем строку как массив символов.
12              //При этом один символ резервируем для терминального ноля.
13              char* str = new char[256];
14              //Чтение всей строки, включая пробелы:
15              cin.getline(str, 256);
16              //Определяем длину введенной строки:
17              length = strlen(str);
18              bool palindrom = true;//Признак палиндрома
19              for (int i = 0; i < length / 2; i++){
20                    //Проверяем пары символов на совпадения
21                    //(первый-последний, второй-предпоследний и т.д. до середины).
22                    if (str[i] != str[length - 1 - i]){
23                          //Если нашлось несовпадение - это не палиндром
24                          palindrom = false;
25                          break;//Дальнейшая проверка бесполезна
26                    }
27              }
28              if (length>0){//Иначе ничего не введено и надо завершать работу
29                    cout << "Это " << ((palindrom) ? "" : "не ");
30                    cout << "палиндром\n";
31              }
32              delete[] str; //Освобождаем память из под строки
33        } while (length>0);
34        system("pause");
35  }
```

2. Дана строка длиной до 64 символов, представляющая беззнаковое целое число в двоичной системе счисления. Написать программу для формирования строки, содержащей шестнадцатеричное представление того же числа.

```
1   #include <iostream>
2   using namespace std;
3   //Функция получения десятичного числа из двоичного
4   unsigned long long Bin2Dec(char* bin){
```

```
5       unsigned long long s = 0;//переменная под результат
6       /*
7               Для перевода пользуемся алгоритмом:
8               Основание системы счисления (2) возводится в
9               степень, равную номеру разряда
10              (нумерация справа-налево начиная с 0) и умножается
11              на значение (0 или 1), находящееся на
12              соответствующей позиции. Суммируя такие произведения,
13              получаем число в десятичной системе счисления.
14              (1101 = 1*2^3 + 1*2^2 + 0*2^1 + 1*2^0 = 8+4+0+1 = 13).
15      */
16      //Получаем значение 2^(номер_старшего_разряда)
17      unsigned long long p = 1LL << (strlen(bin) - 1);
18      //Суффикс "LL" означает, что константа 1 должна
19      //восприниматься как длинное целое значение
20      for (int i = 0; i < strlen(bin); i++){
21              //Вычисляем произведение вида <значение_в_разряде>*2^<№разряда>
22              s += (bin[i] == '1') * p;
23              //Переходим к следующему разряду (сдвиг вправо).
24              p >>= 1;
25      }
26      return s;
27  }
28  void Bin2Hex(char* bin, char* hex, int size){
29      //Получаем десятичное представление числа
30      unsigned long long x = Bin2Dec(bin);
31      //Преобразуем в 16-ричный формат:
32      sprintf_s(hex, size, "%llx", x);
33  }
34  void main(){
35      setlocale(LC_ALL, ".1251");
36      cout << "Введите число в двоичной системе счисления: \n";
37      //По условию длина строки до 64 символов
38      //+ 1 символ резервируем под терминальный ноль
39      char *binS = new char[65];
40      //Вводим строку, содержащую двоичное представление целого числа
41      cin >> binS;
42      //Узнаем размер новой строки с
43      //шестнадцатеричным представлением числа
44      int hexSize = strlen(binS);//Узнаем размер двоичной строки
45      /*Размер новой строки надо уменьшить с учетом того,
46      что каждые 4 двоичные цифры будут заменены одной
47      шестнадцатеричной. Причем, если длина двоичной строки
48      не делится нацело на 4, надо добавить дополнительный разряд.
49      Также не следует забывать добавлять 1 символ
50      для терминального ноля.*/
51      int hsz = hexSize / 4 + (hexSize % 4 != 0) + 1;
52      char *hexS = new char[hsz];
53      //Выполняем преобразование
54      Bin2Hex(binS, hexS, hsz);
55      //Выводим результат
56      cout << binS << "=" << hexS << endl;
57      //Освобождаем память
58      delete[] hexS;
59      delete[] binS;
60      system("pause");
61  }
```

Результат работы программы:

```
Введите число в двоичной системе счисления:
10101010111101000001010111010100010101110101111101000001
10101010111101000001010111010100010101110101111101000001=557a0aea2baf41
```

3. Дана строка, изображающая арифметическое выражение вида «число±число±…±число», где на месте знака операции «±» находится знак «+» или «–» (например, «-14+7.6-2.13+8.77»). Вывести значение данного выражения (вещественное число).

```
1   #include <iostream>
2   using namespace std;
3
4   double Parse(const char* str){
5         //Разбиваем строку на подстроки по знакам "+" или "-".
6         char* cstr = _strdup(str);//Создание дубликата строки.
7         char* pstr = cstr;//Создаем копию указателя на строку.
8         char* context;//Контекст разбиения строки
9         char* part;  //Здесь будут содержаться различные части строки
10        double res = 0;
11        char zn = (str[0]=='-')?'-':'+';//Знак выполняемой операции.
12        while (part =  //Здесь получаем очередную часть строки
13              strtok_s(pstr,  //Строка в первой итерации и NULL - в последующих
14              "+-",  //Символы-разделители
15              &context  //Контекст содержит указатель на оставшуюся часть строки
16              )){//Цикл выполняется пока в строке остаются числа
17              pstr = NULL;//Для продолжения разбора строки
18              //После вызова strtok_s в part содержится очередное число в сроковом формате.
19              //Преобразуем его в тип double:
20              double ch;
21              sscanf_s(part, "%lf", &ch);
22              if (zn == '+'){
23                    res += ch;//Добавляем число к результату…
24              }
25              else {
26                    res -= ch;//…или вычитаем, в зависимости от знака операции
27              }
28              /*Оставшаяся часть строки без символа-разделителя находится в
29              переменной context, причем она содержит указатель на некоторую
30              часть копии исходной строки cstr. Таким образом, мы можем узнать,
31              насколько байт смещена оставшаяся неразобранной часть от начала
32              исходной строки, вычислив разность между указателями context - cstr.
33              Добавив то же смещение (в символах) за вычетом 1 к константному
34              параметру str, содержащему исходную строку, получим знак операции,
35              которую необходимо произвести. Этот символ сохраним в переменной
36              zn для использования на следующей итерации. */
37
38              //Смещение в байтах оставшейся части от начала строки
39              int shift = (context - cstr);
```

```
            //То же смещение, но в символах
            shift /= sizeof(*context);
            //Символ, находящийся перед context ("+" или "-")
            zn = *(str+shift-1);
      }
      delete[] cstr;
      return res;
}
void main(){
      setlocale(LC_ALL, ".1251");
      cout << "Введите строку вида «число±число±...±число»: ";
      char* str = new char[256];
      cin >> str;
      cout << Parse(str) << endl;
      delete[] str;
      system("pause");
}
```

Результат работы программы:

```
Введите строку вида <число±число±…±число>: -14+7,6-2,13+8,77
0.24
```

4. Дана строка S. Удалить из строки S подстроки, совпадающие с S_0. Если совпадающих подстрок нет, то оставить строку S без изменений.

```
#include "Windows.h"
#include <iostream>
#include <cstring>
using namespace std;
//Рекурсивная функция удаления подстрок
void DelSub(char* str, char* substr, char* result){
      //Ищем подстроку в строке
      char* pos = strstr(str, substr);
      //Если подстрока найдена, ее надо удалить.
      if (pos){
            //Копируем первую часть строки (до substr):
            strncpy_s(result,//куда копируем
                  _msize(result)/sizeof(*result),//размер буфера
                   str,  //откуда копируем
                   (pos - str)//сколько символов копируем
                  );
            //Копируем вторую часть строки (после substr):
            //и присоединяем к уже имеющейся части
            strncat_s(result,//куда копируем
      msize(result)/sizeof(*result),//размер буфера
      pos+strlen(substr),  //откуда копируем
      (strlen(str)-(pos-str)-strlen(substr))//сколько символов копируем
                  );
            //Одна подстрока удалена.
            //Создаем копию результата
            char* tmp = new char[strlen(result) + 1];
            strcpy_s(tmp, _msize(tmp) / sizeof(*tmp), result);
            //Вызываем функцию удаления подстроки для оставшейся части
            DelSub(tmp, substr, result);
```

```
30              delete[] tmp;
31          }
32      }
33      void main(){
34          SetConsoleCP(1251);
35          SetConsoleOutputCP(1251);
36          cout << "Введите исходную строку: \n";
37          char* s = new char[256];
38          cin.getline(s, 256);
39          cout << "Введите искомую подстроку: \n";
40          char* s0 = new char[50];
41          cin.getline(s0, 50);
42          char* result = new char[256];
43          DelSub(s, s0, result);
44          cout << "Строка после исключения подстрок: ";
45          cout << result << endl;
46          system("pause");
47      }
```

Результат работы программы:

```
Введите исходную строку:
парадопастьпа
Введите искомую подстроку:
па
Строка после исключения подстрок: радость
```

В. Задания для самостоятельного выполнения

1. Написать программу, которая проверяет, является ли введенное предложение палиндромом, без учета пробелов. Например, палиндромом должны считаться предложения: «а роза упала на лапу азора» или «у дуба буду».
2. В условиях предыдущей задачи добавить исключение знаков препинания, чтобы предложения вида «на в лоб, болван!» или «madam, i’m adam.» также определялись как палиндромы.
3. В условиях предыдущей задачи исключить различие заглавных и строчных букв, так чтобы предложения вида «Ума дай Адаму!» или «Ты, Милок, иди яром: у дороги мина, за ДоРоГоЙ огород, а за ним и город у моря; иди, коли мыт.» определялись как палиндромы.
4. Дано число, содержащее номер телефона вида 89xxxxxxxxx, привести его к строке в формате 8 (9xx) xxx-xx-xx. Предусмотреть возможность ввода номеров меньшей длины и верного их форматирования.

5. Дана строка-предложение на русском языке и положительное число K. Зашифровать строку, выполнив циклическую замену каждой буквы на букву того же регистра, расположенную в алфавите на K-й позиции после шифруемой буквы (например, для K=3 «А» перейдет в «Г», «а» — в «г», «Б» — в «Д», «я» — в «в» и т.д.). Буквы «ё» и «Ё» в алфавите учитывать не нужно. Остальные знаки в предложении не изменять.
6. Дана строка, содержащая полное имя файла, то есть имя диска, список папок (путь), собственно имя и расширение. Выделить из этой строки имя файла (без расширения).
7. Дана строка-предложение. Вывести самое длинное слово в предложении. Если таких слов несколько, вывести первое из них. Словом считать набор символов, не содержащий пробелов и знаков препинания и ограниченный пробелами, знаками препинания или началом/концом строки.
8. Дана строка, содержащая цифры. Заменить каждую отдельную цифру в строке на свое текстовое название («0» на «ноль», «1» на «один» и т.д.).
9. Решить задачу, обратную к задаче 8. Все названия цифр («ноль», «три» и т.д.) заменить соответствующими цифрами.
10. Дано целое положительное число N и строка S. Преобразовать строку S в строку длины N следующим образом: если длина строки S больше N, то отбросить первые символы, если длина строки S меньше N, то в ее начало добавить символы «-».

РЕКОМЕНДУЕМАЯ ЛИТЕРАТУРА

1. Канцедал С.А. Алгоритмизация и программирование : Учебное пособие. — М.: ИД ФОРУМ: НИЦ ИНФРА-М, 2014. — 352 с. // http://znanium.com/bookread.php?book=429576
2. Колдаев В. Д. Основы алгоритмизации и программирования: Учебное пособие / Под ред. Л.Г. Гагариной. — М.: ИД ФОРУМ: ИНФРА-М, 2012. — 416 с. //http://znanium.com/bookread.php?book=336649
3. Немцова Т. И. Программирование на языке высокого уровня. Программир. на языке С++: Уч. пос. /Под ред. Л.Г.Гагариной — М.: ИД ФОРУМ: ИНФРА-М, 2012. — 512 с. // http://znanium.com/bookread.php?book=244875
4. Пахомов Б. И. С/С++ и MS Visual C++ 2010 для начинающих. — СПб.: БХВ-Петербург, 2011. — 728 с. // http://znanium.com/bookread.php?book=351461
5. Культин Н.Б. С/С++ в задачах и примерах. — 2-е изд., перераб. и доп. - СПб.: БХВ-Петербург, 2009. — 349 с. // http://znanium.com/bookread.php?book=356661
6. Голощапов А. Л. Microsoft Visual Studio 2010. — СПб.: БХВ-Петербург, 2011. — 543 с. // http://znanium.com/bookread.php?book=354994
7. Дорогов В. Г., Дорогова Е. Г. Основы программирования на языке С: Учебное пособие / Под общ. ред. проф. Л.Г. Гагариной — М.: ИД ФОРУМ: ИНФРА-М, 2011. — 224 с. // http://znanium.com/bookread.php?book=225634
8. Хабибуллин И.Ш. Программирование на языке высокого уровня С/С++. — СПб.: БХВ-Петербург, 2006. — 499 с. // http://znanium.com/bookread.php?book=356906
9. Полубенцева, М. И. С/С++. Процедурное программирование / М.И. Полубенцева. — СПб.: БХВ-Петербург, 2008. — 414 с. // http://znanium.com/bookread.php?book=350407
10. Стахнов А. А. Linux: 4-е изд., перераб. и доп. — СПб.: БХВ-Петербург, 2011. — 738 с. //http://znanium.com/bookread.php?book=355362
11. Лафоре Р. Объектно-ориентированное программирование в С++.— 4-е изд. — Санкт-Петербург [и др.]: Питер, 2014.— 923 с.
12. Орлов С. А. Теория и практика языков программирования: учебник по направлению "Информатика и вычислительная техника" — Санкт-Петербург [и др.]: Питер, 2014.— 688 с.
13. Тюгашев А. А. Языки программирования: учебное пособие для студентов высших учебных заведений, обучающихся по специальности 10.05.03

(090303) "Информационная безопасность автоматизированных систем" — Санкт-Петербург [и др.]: Питер, 2014.— 333 с.
14. Дронов В. А. HTML 5, CSS 3 и Web 2.0. Разработка современных Web-сайтов. — СПб.: БХВ-Петербург, 2011. — 414 с. // http://znanium.com/bookread.php?book=351455
15. Васильев В. В., Сороколетова Н. В., Хливненко Л. В. Практикум по Web-технологиям. — М.: Форум, 2009. — 416 с. // http://znanium.com/bookread.php?book=166294
16. Беговатов Е.А., Кашина О.А., Лернер Э.Ю. Изучаем законы распределения случайных величин с пакетом Mathematica, 2009.

Учебное пособие

Маклецов Сергей Владиславович

Основы компьютерных наук
Часть 1

Подписано в печать _____.
Бумага офсетная. Печать цифровая.
Формат 60x84 1/16. Гарнитура «Times New Roman». Усл. печ. л. .
Тираж экз. Заказ

Отпечатано с готового оригинал-макета
в типографии Издательства Казанского университета

420008, г. Казань, ул. Профессора Нужина, 1/37
тел. (843) 233-73-59, 233-73-28