МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ УКРАИНЫ
Национальный университет кораблестроения
имени адмирала Макарова

**С. Б. ПРИХОДЬКО, Л. Н. МАКАРОВА,
Т. Г. СМЫКОДУБ**

МЕТОДИЧЕСКИЕ УКАЗАНИЯ
к выполнению лабораторных работ по дисциплине "Основы программирования"
В двух частях
Часть 1

Рекомендовано Методическим советом НУК

Электронное издание
комбинированного использования на DVD-ROM

НИКОЛАЕВ • НУК • 2017

УДК 004.4(076)
ББК 32.973.2я73
П 77

Автори:
С. Б. Приходько, д-р техн. наук, професор;
Л. М. Макарова, канд. техн. наук;
Т. Г. Смикодуб, ст. викладач

Рецензент І. І. Коваленко, д-р техн. наук, професор

Приходько С. Б.
П 77 Методичні вказівки до виконання лабораторних робіт з дисципліни "Основи програмування" : у 2 ч. Ч.1 / С. Б. Приходько, Л. М. Макарова, Т. Г. Смикодуб. – Миколаїв : НУК, 2017. – 59 с.

Наведено завдання для лабораторних робіт, коротке викладення теоретичних відомостей, етапи виконання робіт і контрольні питання.

Методичні вказівки призначені для іноземних студентів І курсу спеціальності 121 "Інженерія програмного забезпечення" (попередня назва "Програмне забезпечення систем"), а також для усіх, хто вивчає основи програмування мовою С++.

**УДК 004.4(076)
ББК 32.973.2я73**

Навчальне видання
ПРИХОДЬКО Сергій Борисович
МАКАРОВА Лідія Миколаївна
СМИКОДУБ Тетяна Георгіївна

МЕТОДИЧНІ ВКАЗІВКИ
до виконання лабораторних робіт з дисципліни "Основи програмування"
У двох частинах
Частина 1

(*російською мовою*)

Комп'ютерне складання та верстання *А. В. Платонова*
Коректор *М. О. Паненко*

Формат 60×84/16. Ум. друк. арк.3,5. Об'єм даних 976 кб.
Тираж 15 прим. Вид. № 37. Зам. № 69.
Видавець і виготівник Національний університет кораблебудування
імені адмірала Макарова
просп. Героїв України, 9, м. Миколаїв, 54025
E-mail : publishing@nuos.edu.ua
Свідоцтво суб'єкта видавничої справи ДК № 2506 від 25.05.2006 р.

ВВЕДЕНИЕ

В Национальном университете кораблестроения имени адмирала Макарова (НУК) дисциплина "Основы программирования" изучается студентами специальности 121 "Инженерия программного обеспечения" (предыдущее название "Программное обеспечение систем") в первом–третьем семестрах. Она относится к циклу профессиональной и практической подготовки бакалавров.

Эти методические указания должны помочь студентам первого курса специальности 121 "Инженерия программного обеспечения" в подготовке и выполнении лабораторных работ по дисциплине "Основы программирования". Они содержат задания для лабораторных работ, краткое изложение теоретических сведений, этапы выполнения работ и контрольные вопросы.

При выполнении каждой лабораторной работы рекомендуется:

– усвоить теоретические сведения;

– разобраться с заданием и этапами выполнения работы;

– выполнить предложенные задания и оформить отчет по работе;

– проверить полноту понимания темы с помощью контрольных вопросов, расположенных в конце каждой работы.

Лабораторную работу необходимо выполнять в соответствии с приведенными этапами выполнения работы. Отчет о лабораторной работе оформляется каждым студентом индивидуально. Отчет должен содержать: название и цель работы; задание (постановку задачи); методику и алгоритм решения задачи; текст программы; результаты работы; анализ результатов и выводы.

На вводном занятии студенты знакомятся с работой в NetBeans IDE – интегрированной среде разработки (Integrated Development Environment – IDE) приложений, со следующего занятия начинается выполнение лабораторных работ.

РАБОТА № 1

Разработка и реализация программы с линейной структурой

Цель работы: закрепление знаний алфавита языка программирования C++, приобретение навыков записи его констант, переменных, выражений, операторов присваивания; овладение навыками составления программы с линейной структурой и выполнения ее в NetBeans IDE.

Задания

Задание 1.1. Записать на языке C++ математические выражения по вариантам, которые приведены в таблице 1.1.

Таблица 1.1 – Варианты задания 1.1

№	Математические выражения	№	Математические выражения
1–3	а) $3(z+1)^2+2,1\cdot10^6$ б) $\lvert x+z\rvert>0 \wedge 0<b<1$	4–6	а) $10^{-4}\ e^{-2f}+\lvert z^3\rvert$ б) $x+z<0 \vee 0<f<2$
7–9	а) $3(z+3)^{0,5}+10^{-3}$ б) $\lvert x+z\rvert>1 \wedge 1<b<2$	10–12	а) $\ln(z+1)^{0,5}+4,2\cdot10^4$ б) $\lvert x\rvert>2 \vee 0<b<3$
13–15	а) $5(z+1)^5+10^6$ б) $0<b<1 \vee 0<f<0,5$	16–18	а) $6(z+1)^6+1,6\cdot10^6$ б) $\cos\lvert x+z\rvert>0 \vee 0<b<6$
19–21	а) $7(z+1)^{0,5}+2,7\cdot10^6$ б) $\lvert x+z\rvert>0 \wedge 0<b<7$	22–24	а) $10^{-7}\sin\lvert 3z\rvert+b^{0,5}$ б) $\ln\lvert x+z\rvert>0 \vee 0<b<1$
25–27	а) $\lvert z+1\rvert^3+2\cdot10^5$ б) $\ln\lvert x+z\rvert>0 \wedge b>9$	28–30	а) $(z+1)^{0,5}+10^6$ б) $\lvert x+z\rvert>0 \vee 0<b<1$

Принятые обозначения: $\vee$ – объединение множеств, $\wedge$ – пересечение множеств.

Задание 1.2. Представить математическую запись выражения по вариантам, которые приведены в таблице 1.2, и показать порядок действий.

Таблица 1.2 – Варианты задания 1.2

№	Выражение
1	`x+2./3./x/a+sin(x)/2*sqrt(x)+1.0e-6*pow(x,1./7.)`
2	`(x+7)/3.*x+3*sqrt(x)/2./x+1.0e7-1./3.*pow(x,5)`
3	`x+2./3.*x/a+cos(x)/2./sqrt(x)+1.0e-5*pow(x,7)`
4	`(x+4)/3./x+sqrt(abs(x))/2.*x+1.0e-6*pow(x,1./3.)`
5	`x+2./3./x/a+sqrt(x)/2./sin(x)+1.0e5*pow(x/3.,2./7.`
6	`1.4e-4*pow(2*x,3)+sqrt(sin(x))/2.+cos(x)/2./x`
7	`abs(cos(x))/2./x-5./7.*x/a/1.0e-6*pow(x/2.,1./7.)`
8	`x+2./3./x*a+sqrt(sin(x))/2/x+1.0e-3*pow(x/7.,2./3.`
9	`(x+7)/3.*x+3*sqrt(x)/2./x+1.0e7-4*pow(x,b)`
10	`sqrt(cos(x*x))/2./x-5./7.*x/a/1.0e-6*pow(x,1./8.)`
11	`x+9./(3*x/a)+cos(x)/2./sqrt(x)+1.0e-5*pow(x,9)`
12	`x+4./3./x+exp(abs(x))/2.*x+1.0e-4*pow(x,1./3.)`
13	`sqrt(x)/2./(x-5./7.)*x/a/1.0e-6*pow(x/2.,1./9.)/11`
14	`x+2*3./x*a+sin(x)/(2*x+1.0e-3*pow(x,5./7.)`
15	`x+4./3./(x+abs(x))/2.*x+1.0e-5*pow(x,5./3.)`

Задание 1.3. Составить программу вычисления следующих величин по вариантам, которые приведены в таблице 1.3, и выполнить ее в NetBeans IDE.

Таблица 1.3 – Варианты задания 1.3

№	Величины, которые необходимо вычислить
1	Модуль вектора $5\boldsymbol{a}+10\boldsymbol{b}$, если $\boldsymbol{a}=\{3; 2\}$ и $\boldsymbol{b}=\{0; -1\}$
2	Проекция вектора $\boldsymbol{a}=\{5; 2; 5\}$ на ось вектора $\boldsymbol{b}=\{2; -1; 2\}$
3	Периметр треугольника с вершинами A(1; 1), B(4; 1), C(4; 5)
4	Модуль вектора $-2\boldsymbol{a}+4\boldsymbol{b}$, если $\boldsymbol{a}=\{3; 2\}$ и $\boldsymbol{b}=\{0; -1\}$

Продолж. табл. 1.3

5	Углы треугольника с вершинами A(0; 1,7), B(2; 1,7), C(1,5; 0,85)
6	Площадь четырехугольника с вершинами A(0;0), B(–1;3), C(2;4), D(3;1)
7	Модуль вектора $\boldsymbol{a}+\boldsymbol{b}$, если $\lvert\boldsymbol{a}\rvert=11$, $\lvert\boldsymbol{b}\rvert=23$, $\lvert\boldsymbol{a}-\boldsymbol{b}\rvert=30$
8	Модуль вектора $\boldsymbol{a}-\boldsymbol{b}$, если $\lvert\boldsymbol{a}\rvert=3$, $\lvert\boldsymbol{b}\rvert=5$, и $\boldsymbol{a}$ и $\boldsymbol{b}$ образуют угол в 120^0
9	Угол между векторами $\boldsymbol{a}=\{2; -4; 4\}$ и $\boldsymbol{b}=\{-3; 2; 6\}$
10	Модуль вектора $\boldsymbol{a}+\boldsymbol{b}$, если $\boldsymbol{a}=\{1; 2; 3\}$ и $\boldsymbol{b}=\{4; -2; 9\}$

Краткие теоретические сведения

Алфавит языка C++ состоит из совокупности символов, которые приведены в приложении А. С помощью символов алфавита можно составлять различные конструкции: константы, переменные, операторы, и т. д. Управляющие символы языка C++, или escape-последовательности, приведены в приложении Б.

Постоянная величина (константа) не изменяется в процессе работы программы. Есть несколько видов констант: числовые (целые и с плавающей точкой), символьные, логические.

Целые десятичные константы состоят из последовательности десятичных цифр, перед которой нет цифры 0, а может стоять знак "+" или "–". Пример: `-15`.

Целые восьмеричные константы состоят из последовательности восьмеричных цифр, перед которой есть цифра 0 и может стоять знак "+" или "–". Пример: `-015`.

В языке C++ целые константы можно изображать и в шестнадцатеричной системе счисления. В этом случае константы начинаются с `0x` или `0X`, за которыми следуют шестнадцатеричные цифры. Перед знаком `0x` или `0X` может быть знак "+" или "–". Пример: `0x2A`.

Целые десятичная, восьмеричная и шестнадцатеричная константа, значение которой превышает наибольшее машинное целое со знаком, является длинной константой (**long**); в других случаях целые константы считаются **int**.

Целые десятичная, восьмеричная и шестнадцатеричная константа, за которой непосредственно стоит буква `l` или `L`, является длинной константой (**long**).

Константа с плавающей точкой состоит из целой части, десятичной точки, дробной части, буквы `e` или `E` и целого показателя степени. Целая или дробная часть (но не обе сразу) может отсутствовать. Десятичная точка, или `e` (`E`) совместно с целым показателем степени (но не обе части одновременно) может отсутствовать. Например, число 0,015 можно записать как `0.015` и `0.15E-01`, или `1.5E-02`. Константа с плавающей точкой имеет тип **double**. Параметры форматирования потока для ввода/вывода подробнее приведены в приложении З.

Символьная константа – это символ в кавычках. Пример: `'f'`. Символьные константы считаются данными типа **int**.

Логические константы принимают только два значения: **true** (истина) и **false** (ложь).

Комментарий – это текст, ограниченный символами `/ *` и `* /`, или текст, который начинается символами `//` и заканчивается в конце строки. Комментарии `/ * * /` не могут быть вложенными. Символы `//` можно использовать для того, чтобы закомментировать символы `/ *` или `* /`, а символами `/ *` можно закомментировать //.

Переменная обозначается именем (идентификатором), типом и значением. Идентификатор является последовательностью букв и цифр, причем первый символ должен быть буквой. Символ подчеркивания "_" считается буквой. В идентификаторах большие и малые буквы различаются, буквы кириллицы использовать нельзя. Нельзя использовать в качестве идентификаторов служебные слова, список которых приведен в приложении В.

Язык С++ является строго типизированным языком. Это означает, что каждая переменная должна быть описана, то есть необходимо определить ее тип.

Обработка данных выполняется в выражениях. Выражение состоит из операций и операндов. Операндом может быть константа, переменная или имя функции (математические функции приведены в приложении Г). Вычисления выполняются слева направо с учетом приоритета операций. Операции по приоритету с описанием их обозначений, названий и синтаксиса приведены в приложении Д.

Отметим, что при выполнении операции деления "/" целого на целое в результате получаем целую долю. Например, `5/2` равно `2`.

Префиксные операции расположены слева от операнда и выполняют действие с ним до того, как его значение будет использовано, а постфиксные операции выполняются после всех вычислений.

Логические операции `!`, `&&`, и `||` осуществляют действия над операндами в соответствии с таблицей истинности (см. табл. 1.4). Например, при `k = 1` результатом выражения `0>k && 4<k` является `true`.

Таблица 1.4 – Таблица истинности логических операций

Операция	Действие	Выражение	A	B	Результат
!	Логическое НЕТ ¬	!A	true false		false true
&&	Логическое И ∧	A && B	true true false false	true false true false	true false false false
||	Логическое ИЛИ ∨	A || B	true true false false	true false true false	true true true false

Программа на языке C++ состоит из следующих блоков: блока заголовков программы; блока с объявлением классов, прототипов и функций; главного метода программы; блока с описанием функций.

В блоке заголовков программы подключаются внешние файлы, задается область имен, приводятся директивы препроцессору и указания компилятору.

Для подключения к программе внешних файлов (стандартных библиотек или пользовательских файлов) необходимо использовать директиву `#include` с указанием имени подключаемого файла. Директива `#include` включает содержимое файла, путь к которому задан, в компилируемый файл вместо строки с директивой. Если путь помещен в угловые скобки, то поиск файла осуществляется в стандартных директориях. Если путь помещен в кавычки и задан полностью, то поиск файла осуществляется в заданной директории, а если путь полностью не задан – в текущей директории. Некоторые стандартные библиотеки приведены ниже:

`iostream` – библиотека ввода-вывода;

`cmath` – математическая библиотека;

`cstring` – библиотека работы со строками.

Для задания области имен используется директива `using namespace` с указанием конкретной области имен. Для использования стандартной области имен необходимо использовать конструкцию `using namespace std`;

Заголовки могут меняться в зависимости от используемого компилятора.

Главный метод программы имеет стандартное название `main` (). Выполнение программы начинается с первого оператора этой функции. Хотя `main` () и не является ключевым словом, относиться к нему следует как к ключевому, например, не следует использовать его как имя переменной.

Блок с описанием функций содержит описание тех функций, прототипы которых указаны во втором блоке.

Программа может состоять из нескольких модулей (входных файлов).

Организация вывода данных в консоль выполняется с помощью команды `cout<<`, при этом необходимо подключить библиотеку `iostream` и пространство имен, к которому принадлежит команда `cout<<: using namespace std`. Строку вывода записывают в двойные кавычки. Организация ввода данных с консоли выполняется с помощью команды `cin>>`.

Пример выполнения работы

Задание 1.1. Данные математические выражения записать на языке С++:

а) $10^{-3} \cdot e^{-2x} + 5(z+1)^3$ б) $|b| < 0 \wedge f > 1$.

Решение

а) `1.0e-3*exp(-2*x)+5*pow(z+1, 3)`

б) `abs(b)<0 && f>1`

Задание 1.2. Представить математическую запись выражения `0.51e-6*sqrt(3*x)+2*b*b` и показать порядок действий.

Решение

$0{,}51 \cdot 10^{-6} \cdot (3x)^{0,5} + 2 \cdot b^2$ `0.51e-6*sqrt(3*x)+2*b*b`

3 2 1 6 4 5

Задание1.3. Составить программу вычисления скалярного произведения двух векторов a = {4; –2; –4} и b = {6; –3; 2} и выполнить ее в NetBeans IDE.

Решение

Составить программу вычисления скалярного произведения двух векторов a = {4; –2; –4} и b = {6; –3; 2} на языке С++.

2. Методика решения задачи

Скалярное произведение двух векторов вычисляется по формуле:

$$\boldsymbol{Ab} = A_1 \cdot B_1 + A_2 \cdot B_2 + A_3 \cdot B_3, \qquad (1.1)$$

где A_1, B_1, A_2, B_2, A_3, B_3 – соответствующие координаты векторов ***a*** и ***b***.

3. Алгоритм решения задачи

Алгоритм решения задачи можно представить в виде такой последовательности действий:

Действие 1. Ввести координаты векторов ***a*** и ***b***.

Действие 2. Вычислить скалярное произведение по формуле (1.1).

Действие 3. Вывести значение скалярного произведения двух векторов.

Представим алгоритм решения задачи на языке C++, обозначив переменные A_1, B_1, A_2, B_2, A_3, B_3 и ***ab*** соответственно как `a1, b1, a2, b2, a3, b3` и `ab` (все типа **`double`**).

4. Текст программы

```
// Вычисление скалярного произведения двух
векторов
#include <iostream>
#include <cmath>
using namespace std;
int main(){
   double a1, a2, a3, b1, b2, b3, ab;
   cout <<"Введите координаты a1,a2,a3,b1,b2,b3"<<endl;
   cin>>a1>>a2>>a3>>b1>>b2>>b3;
   ab=a1*b1+a2*b2+a3*b3;
   cout<<"Скалярное произведение ab="<<ab<<endl;
}
```

5. Результаты работы программы

```
Введите координаты a1,a2,a3,b1,b2,b3
4
-2
-4
6
```

```
-3
2
Скалярное произведение ab=22
```

Контрольные вопросы

1. Какие данные можно использовать в языке C++?
2. С какой целью используют директиву `#include`?
3. Назовите порядок выполнения операций в выражении.
4. Какова структура программы на языке C++?
5. Как работает оператор присваивания?
6. Как в C++ осуществляется ввод и вывод значений переменных?

РАБОТА № 2

Разработка и реализация программы с разветвленной структурой

Цель работы: овладение навыками составления программы с разветвленной структурой с помощью условного оператора **if** или оператора выбора **switch** и выполнение ее в NetBeans IDE.

Задания

Задание 2.1. Представить математическую запись фрагмента программы по вариантам, которые приведены в таблице 2.1, и вычислить значение переменной x после его выполнения. Указание: вместо n подставить номер варианта.

Таблица 2.1 – Варианты задания 2.1

№	Фрагмент программы	№	Фрагмент программы

1–5	`t=n; x=t;`
	`if(t>1 && t<3)x=3;`
	`if(t<=1) x=0;`
		6–10	`t=n; x=0;`
			`if(t<0)x=-t;`
			`else x=t;`

11–15	`a=n; b=13; c=12; x=a;`
	`if(x<b)x=b;`
	`if(x<c)x=c;`
		16–20	`a=n; b=17; c=18; x=a;`
			`if(b<x)x=b;`
			`if(c<x)x=c;`

21–25	`t=n;`
	`switch(t){`
	`case 0: x=1; break;`
	`default: x=0;};`
		26–30	`t=n;`
			`if(t>0 && t<28)x=10;`
			`else if(t>=28)x=20;`
			`else x=0;`

Задание 2.2. Составить программу вычисления значений функции по вариантам, которые приведены в таблице 2.2, и выполнить ее в NetBeans IDE.

Таблица 2.2 – Варианты задания 2.2

№	Функция	№	Функция	№	Функция	№	Функция
1	$y = \mathrm{tg}x/\ln x$	2	$y = \ln x/\mathrm{tg}x$	3	$y = \arcsin(1/x)$	4	$y = \mathrm{ctg}\ \ln x$
5	$y = \mathrm{ctg}\, x^{1/3}$	6	$y = \mathrm{tg}x/(1-x)$	7	$y = \mathrm{tg}x/\ln^{2/3} x$	8	$y = x^{0,2}\mathrm{tg}x$
9	$y = \ln \mathrm{tg}\, x$	10	$y = \mathrm{tg}^{1/3} x/(x+1)$	11	$y = \mathrm{arcctg}^{1/3} x$	12	$y = \arccos x$
13	$y = \arcsin x$	14	$y = \mathrm{tg}x/(x-1)$	15	$y = \ln x/(1-x^2)$	16	$y = x/(1+\mathrm{tg}x)$
17	$y = \ln x/(1-x)$	18	$y = \mathrm{arcctg}\, x$	19	$y = \mathrm{tg}x/\ln x$	20	$y = x^{1/7}\mathrm{ctg}x$
21	$y = \arccos(1/x)$	22	$y = x^{1/3}/(1-x^2)$	23	$y = \arcsin^{1/3} x$	24	$y = \mathrm{arcctg}\, x^{1/3}$
25	$y = \mathrm{tg}^2 \ln x$	26	$y = \mathrm{tg}^{1/3} x$	27	$y = \mathrm{tg}x/(1-x^2)$	28	$y = \ln \mathrm{tg}x^{1/3}$

Краткие теоретические сведения

Условный оператор **`if`** предназначен для выполнения или невыполнения некоторого оператора (простого или составного) в зависимости от значения выражения.

Общие виды условного оператора **`if`**:

`if`(*выражение*) *оператор1*;

`if`(*выражение*) *оператор1* **`else`** *оператор2*;

Выполнение условного оператора **`if`** заключается в вычислении логического *выражения*. Если его значения **`true`** (не равно нулю), то выполняется *оператор1*. Если значения логического выражения **`false`** (равно нулю) и условный оператор не содержит слова **`else`**, то его выполнение завершается. Если слово **`else`** присутствует, то выполняется *оператор2*.

Будьте внимательны при записи логического *выражения*: часто вместо операции проверки на равенство == указывается операция присваивания =, что приводит к некорректной работе программы. Служебные слова **`if`** и **`else`** неразрывны, при попытке вставить между ними любую команду возникает ошибка на этапе компиляции.

Если в какой-либо ветви условного оператора надо выполнить несколько операторов, то их следует объединить в составной оператор с помощью операторных скобок `{}`. Один условный оператор может входить в другой условный оператор. При этом каждое слово **`else`** соответствует последнему перед ним **`if`**. Так, после выполнения следующего фрагмента программы:

```
int y, x=3;
if (x>0 && x<=1)y=1;
else if (x>1)y=10;
     else y=0;
```

переменная `y` имеет значение `10`.

Оператор выбора **`switch`** предназначен для выполнения одного из нескольких возможных операторов в зависимости от совпадения значения выражения-селектора и значения константного выражения. Общий вид оператора **`switch`**:

`switch` (*выражение-селектор*) {
    `case` *константное выражение 1* : *оператор1*; **`break`**;
        . . .
    `case` *константное выражение N* : *операторN*; **`break`**;
    `default`: *оператор*;
}

Выражение-селектор должен быть целого типа, типа **`char`** или типа указателя. *Константное выражение* должно иметь такой же тип, что и *выражение-селектор*. В одном операторе **`switch`** никакие *константные выражения* не могут иметь одинаковых значений. Может также быть не более одного префикса **`default`** или он может отсутствовать.

Выполнение оператора **`switch`** начинается с вычисления значения *выражения-селектора*. При первом совпадении этого значения со значением *константного выражения* (1, ..., N) выполняется соответствующий *оператор*. Если ни одного

совпадения не зафиксировано, а есть слово **default**, то выполняется *оператор*, следующий за **default**. В противном случае выполнения оператора **switch** завершается.

Следует отметить, что префиксы **case** и **default** сами не изменяют поток управления. Для выхода из оператора **switch** применяется оператор **break**.

Оператор **break** прекращает выполнение оператора **switch**, в котором был выполнен этот оператор. Управление передается на оператор, который стоит сразу за этим оператором **switch**.

В случае, когда необходимо выполнить одинаковые действия для различных значений *константных выражений*, можно записывать несколько выражений подряд.

Пример. После выполнения следующего фрагмента программы:

```
char letter= 'A';
switch (letter){
 case 'A': cout<<"Case A"; break;
 case 'B':
 case 'C': cout<<"Case B or C"; break;
 default: cout<<"Not A, B or C";
}
```

будет напечатано `Case A`

Пример выполнения работы

Задание 2.1. Представить математическую запись фрагмента программы

```
t=-8.;
if(t>0)x=pow(t,1./3.);
else  if(t<0)  x=-pow(abs(t),1./3.);
      else x=0;
```

и вычислить значение переменной x после его выполнения.

Решение

Этот фрагмент программы реализует вычисление функции $x=t^{1/3}$. После выполнения этого фрагмента $x=-2$.

Задание 2.2. Составить программу вычисления значений функции $y=\mathrm{ctg}x$ и выполнить ее в NetBeans IDE.

Решение

1. Постановка задачи

Составить программу вычисления значений функции $y=\mathrm{ctg}x$ на языке C++.

2. Методика решения задачи

Функция $y=\mathrm{ctg}x$ вычисляется по формуле

$$y = \cos x/\sin x, \qquad (2.1)$$

если

$$\sin x \neq 0. \qquad (2.2)$$

3. Алгоритм решения задачи

Алгоритм решения задачи можно представить в виде такой последовательности действий:

Действие 1. Ввести значение x.

Действие 2. Проверить условие (2.2). Если условие истинно, то вычислить значение функции по формуле (2.1) и вывести его, иначе вывести сообщение: "Функция не существует".

Окончательно представим алгоритм решения задачи на языке C++, обозначив переменные x и y соответственно как x и y (обе типа **double**).

4. Текст программы

```
// программа вычисления функции y=ctg(x)
#include <iostream>
#include <cmath>
using namespace std;
int main(){
   double x,y;
   cout<<"Введите x="; cin>>x;
```

```
    if(sin(x)!=0){
      y=cos(x)/sin(x);
      cout<<"y="<<y<<endl;
      }
    else cout<<"Функция не существует"<<endl;
}
```

5. Результаты работы программы

```
Введите  x=1.57
y=0.000796327
```

Контрольные вопросы

1. Как работают операторы **if** и **switch**?
2. Когда в операторе **if** применяют составной оператор?
3. Как выполняется условный оператор **if**, если в него входит другой условный оператор **if**?
4. Для чего предназначен оператор **switch**?
5. Какого типа может быть выражение-селектор в операторе **switch**?
6. Что делает оператор **break** в случае его выполнения в операторе **switch**?

РАБОТА № 3

Разработка и реализация программы с циклической структурой

Цель работы: овладение навыками составления программы с циклической структурой с помощью операторов цикла **while, do while** и **for** и выполнения ее в NetBeans IDE.

Задания

Задание 3.1. Представить математическую запись фрагмента программы по вариантам, которые приведены в таблице 3.1, и вычислить значение переменной x после его выполнения. Указание: вместо n подставить номер варианта.

Таблица 3.1 – Варианты задания 3.1

№	Фрагмент программы	№	Фрагмент программы

1-5	x=1;
	for(j=n; j>n; j--)
	  x=x*j;
	x=2*x;
		6-10	x=0; j=1;
			do
			  x=x+j; j=j+2;
			while(j<=n);

11-15	x=0;
	for(j=1; j<=n; j++)
	  x=x+2;
	x=2*x;
		16-20	x=1;
			while(x<=n)
			  x=x+1;
			x=2*x;

21-25	x=1; j=1; x1=n%5;
	do
	  x=x*x1; j=j+1;
	  if(j==4) **break**;
	while(j<=5);
		26-30	x=n;
			for(k=1; k<=6; k++){
			  if(k<3) **continue**;
			  x=x+1;
			};

Задание 3.2. Составить программу табулирования функции из задания 2.2 при изменении значения x от –1 до 1 с шагом 0,2 и выполнить ее в NetBeans IDE.

Краткие теоретические сведения

Оператор цикла с предусловием **while** состоит из ключевого слова **while**, за которым следуют *выражение* логического типа в круглых скобках и исполняемый в цикле *оператор* (простой или составной).

Общий вид оператора цикла с предусловием:

while (*выражение*) *оператор – тело цикла*;

Выполнение оператора цикла с предусловием начинается с вычисления значения *выражения*. Если это значение **false**, то *оператор* не выполняется, управление передается на оператор, который стоит сразу за циклом. Если значение *выражения* **true**, *оператор* выполняется, после чего снова вычисляется *выражение*. Чтобы предотвратить зацикливание, следует предусмотреть изменение значения *выражения* внутри *тела цикла*. Например, после выполнения следующего фрагмента программы:

```
bool a=true; int x=5;
while(a || x<9){//цикл завершится, как только
   a=!a; x=x+2;//выражение примет значение false
}
```

переменная x имеет значение 11, а переменная a – 0 (**false**).

Оператор цикла с постусловием **do while** состоит из ключевого слова **do**, за которым следуют исполняемый в цикле *оператор* (простой или составной); ключевого слова **while** и *выражения* логического типа.

Общий вид оператора цикла с постусловием:

do *оператор – тело цикла* **while** (*выражение*);

Выполнение этого оператора цикла происходит так. Сначала выполняется *оператор*, а затем определяется значение

выражения логического типа. Если значение *выражения* **false**, то выполнение цикла прекращается. Если это значение **true**, то выполняется *оператор*, а затем снова вычисляется *выражение*. Например, после выполнения следующего фрагмента программы:

```
bool a=true; int x=5;
do {
a=!a; x=x+2;}// цикл завершится, как только
while(a||x<9);//выражение примет значение false
```

переменная x имеет значение 11, а переменная a – 0 (**false**).

Надо подчеркнуть, что в отличие от цикла **while**, тело цикла с постусловием **do while** всегда выполняется хотя бы один раз.

Оператор цикла с параметром **for** состоит из ключевого слова **for**, за которым следуют в круглых скобках *выражение1*, точка с запятой (**;**), *выражение2*, точка с запятой (**;**), *выражение3* и исполняемый в цикле оператор (простой или составной).

Общий вид оператора цикла **for**:

for (*выражение1*; *выражение2*; *выражение3*) *оператор – тело цикла*;

Выражение1 – это выражение, описывающее инициализацию цикла; *выражение2* – проверка условия завершения цикла; *выражение3* – это выражение, которое вычисляется после каждой итерации цикла. Каждое из *выражений1–3* может состоять из нескольких выражений, которые объединяются оператором запятая (**,**). Ни одно из *выражений1–3* не является обязательным.

Выполнение оператора цикла **for** начинается с вычисления *выражения1*. Затем вычисляется *выражение2*. Если значение *выражения2* **false,** то выполнение цикла прекращается и управление передается на оператор, который стоит сразу за циклом. Если это значение **true**, то последовательно

выполняются *оператор* и *выражение3*, а затем снова вычисляется *выражение2*.

Оператор цикла **for** эквивалентен следующей последовательности операторов:

выражение1;
while (*выражение2*) {
  оператор; выражение3;
}

Если нет *выражения2*, то считается, что он имеет значение **true**. Оператор **for** (;;); представляет собой бесконечный цикл, который эквивалентен оператору **while**(**true**);.

Пример. После выполнения следующего фрагмента программы:

```
bool a; int x;
for(a=true, x=5; a || x<9; a=!a, x+=2);
```

переменная x имеет значение 11, а переменная a – 0 (**false**).

Оценка погрешности при работе с действительными числами на цифровом компьютере, а также способы ее уменьшения приведены в Приложении 3.

Оператор **break** прекращает выполнение оператора цикла (**while**, **do while** или **for**), в котором был выполнен этот оператор. Управление передается на оператор, который стоит сразу за оператором цикла.

Оператор **continue** прекращает выполнение текущей итерации оператора цикла (**while**, **do while** или **for**), в которой был выполнен этот оператор, и осуществляет переход к выполнению следующей итерации цикла. Один оператор цикла может входить в другой оператор цикла.

Пример выполнения работы

Задание 3.1. Представить математическую запись фрагмента программы

```
for(x=1,j=1; j<5; j++, x*=j);
```

и вычислить значение переменной x после его выполнения.

Решение

Этот фрагмент программы реализует вычисления $x!=1\cdot2\cdot...\cdot5$. После выполнения этого фрагмента x=120.

Задание 3.2. Составить программу табулирования функции $y = \mathrm{ctg}x$ при изменении значения x от a=–1 до b=1 с шагом h=0,5 и выполнить ее в NetBeans IDE.

Решение

1. Постановка задачи

Составить программу табулирования функции $y = \mathrm{ctg}x$ при изменении значения x от a=–1 до b=1 с шагом h=0,5 на языке C++.

2. Методика решения задачи

Методика решения задачи совпадает с методикой из примера задачи 2.2 (при оформлении работы необходимо привести методику заново).

3. Алгоритм решения задачи

Алгоритм решения задачи можно представить в виде такой последовательности действий:

Действие 1. Ввести значения a, b, h.

Действие 2. Присвоить x значение a.

Действие 3. Повторять следующие действия пока $x \le b$:

Действие 3.1. Проверить условие (2.2). Если условие истинно, то вычислить значение функции y по формуле (2.1) и вывести x и y, иначе вывести x и сообщение: "не существует";

Действие 3.2. Присвоить x новое значение, равное старому значению x плюс шаг h.

Запишем алгоритм решения задачи на языке C++, обозначив переменные x, y, a, b, h соответственно как `x, y, a, b, h` (все типа **`double`**).

4. Текст программы

```
#include <iostream.h>
#include <cmath.h>
using namespace std;
void main(){
```

```
   double a,b,h,x,y;
   cout<<"Введите a,b,h: "; cin>>a>>b>>h;
   for(x=a; x<=b; x+=h)
     if(sin(x)!=0){
     y=cos(x)/sin(x); cout<<"x="<<x<<"
y="<<y<<endl;
   } else cout<<"x="<<x<<" y="<<"не
существует"<<endl;
}
```

5. Результаты работы программы

```
Введите a,b,h: -1
1
0.5
x=-1  y=-0.642093
x=-0.5  y=-1.83049
x=0  y= не існує
x=0.5  y=1.83049
x=1  y=0.642093
```

Контрольные вопросы

1. Как работают операторы цикла **while**, **do while** и **for**?
2. Чем цикл **do while** отличается от цикла **while**?
3. Когда в циклах **do while** и **for** применяют составной оператор?
4. Какие обязательные элементы присутствуют в цикле **for**?
5. Какой последовательности операторов эквивалентен оператор цикла **for**?
6. Как работают операторы **break** и **continue**?

РАБОТА № 4

Разработка и реализация программы с массивами

Цель работы: овладение навыками составления программы с массивами и выполнения ее в NetBeans IDE.

Задания

Задание 4.1. Определить действие фрагмента программы по вариантам, которые приведены в таблице 4.1, если `char sName[20]="Hello World!"; char *p=sName; int n,i;` (Указание: n равен номеру варианта).

Таблица 4.1 – Варианты задания 4.1

№	Фрагмент программы	№	Фрагмент программы
1–5	`for(p+=n; *p; p++) cout<<*p<<" ";`	6–10	`for(p+=n;p!=sName-1;p--) cout<<*p<<" ";`
11–15	`for(p+=n-9; *p; p++) cout<<*p<<endl;`	16–20	`for(p+=n-11;p!=sName-1;p--) cout<<*p<<endl;`
21–25	`for(i=n-21; i<5; i++) cout<<sName[i];`	26–30	`for(i=n-21; i>2; i--) cout<<sName[i];`

Задание 4.2. Составить программу вычисления следующих величин по вариантам, которые приведены в таблице 4.2, и выполнить ее в NetBeans IDE, если элементы массива определяются по формуле $a_{i+1}=(37\cdot a_i+3) \bmod 64$. Значение a_0 равно номеру варианта; i изменяется от 0 до 18.

Таблица 4.2 – Варианты задания 4.2

№	Величины, которые необходимо вычислить
1–3	Наибольший элемент массива ***a*** и его порядковый номер
4–6	Суммы элементов массива ***a***, значения которых кратны номеру варианта
7–9	Суммы элементов массива ***a***, значения которых четные числа
10–12	Среднее арифметическое положительных элементов массива ***a***
13–15	Суммы элементов массива ***a***, значения которых нечетные числа
16–18	Среднее геометрическое положительных элементов массива ***a***
19–21	Суммы элементов массива ***a***, значения которых двузначные четные числа
22–24	Произведение наибольшего и наименьшего элементов массива ***a***
25–27	Суммы элементов массива ***a***, значения которых двузначные нечетные числа
28–30	Модуль вектора ***a***/3

Краткие теоретические сведения

Массив – это упорядоченная последовательность одноименных элементов, каждый из которых имеет один и тот же тип. Элементы массива могут иметь любой стандартный тип и тип, введенный пользователем. Элементы массива расположены упорядоченно, каждый имеет свой номер, который называется индексом. Доступ к элементам массива осуществляется путем указания имени массива и порядкового номера элемента (индекса).

Массив определяется модификатором типа [ ]. Общий вид описания массива:

тип_элементов имя_масива[количество_элементов];

Например:

```
char  sName[20];
```

определяет массив из двадцати элементов типа **char**. Имя массива `sName` содержит адрес первого элемента массива. Это имя может быть использовано в операциях арифметики указателей для доступа к элементам массива.

Оператор [ ] дает более краткое выражение для доступа к элементам массива. Выражение `sName[k]` эквивалентно `*(sName+k)`. Более подробно работа с оператором * (разыменования) будет описана в части 2 Методических указаний.

Массивы в С++, как и в С, нумеруются с нуля. Поэтому наибольший элемент – это *количество_элементов* минус единица. В массиве из 20 элементов индексы изменяются от 0 до 19. Переход через максимум приводит к использованию чужой области памяти и ошибки на этапе выполнения программы.

Элементы массива размещаются в памяти последовательно. Элементы с меньшими значениями индекса хранятся в более низких адресах памяти. Элементы многомерных массивов размещаются таким образом, что самый правый индекс растет самым первым. Так, для двумерного массива `int d[2][2]` элементы массива размещаются в памяти по возрастанию адресов: `d[0][0], d[0][1], d[1][0], d[1][1]`.

Операции с массивами удобнее проводить с помощью циклов. Невозможно присвоить один массив другому целиком, только поэлементно.

Пример выполнения работы

Задание 4.1. Определить действие фрагмента программы

```
char sName[20]="Hello!"; char *p=sName; int n,i;
for(i=1; i<5; i++) cout<<sName[i];
```

Решение

Этот фрагмент программы выводит на экран строку: `ello`

Задание 4.2. Составить программу перестановки элементов массива a в обратном порядке и выполнить ее в NetBeans IDE, если элементы массива определяются по формуле $a_{i+1}=(37\cdot a_i+3) \bmod 64$. Значение $a_0=40$; i изменяется от 0 до 16.

Решение

1. Постановка задачи

Составить программу перестановки элементов массива ***a*** в обратном порядке на языке С++, если элементы массива

определяются по формуле $a_{i+1}=(37\cdot a_i+3) \bmod 64$. Значение $a_0=40$; i изменяется от 0 до 16.

2. Алгоритм решения задачи

Алгоритм решения задачи можно представить в виде такой последовательности действий:

Действие 1. Ввести элементы массива ***a***.

Действие 2. Вывести элементы массива ***a***.

Действие 3. Определить *m* (количество перестановок элементов массива ***a***), как результат целочисленного деления числа элементов массива *n* на 2.

Действие 4. Присвоить *j* (номеру текущего элемента массива, который переставляется на место элемента с меньшим номером) значение *n*–1.

Действие 5. Повторять *m* раз следующие действия (*i*= 0, 1, ..., *m*-1):

Действие 5.1. Присвоить *x* (переменной для временного хранения элемента с меньшим номером) значение *i*-го элемента массива ***a***;

Действие 5.2. Присвоить *i*-му элементу массива ***a*** значение *j*-го элемента;

Действие 5.3. Присвоить *j*-му элементу массива ***a*** значение *x*;

Действие 5.4. Уменьшить значение *j* на 1;

Действие 6. Вывести элементы массива ***a*** после перестановки.

Запишем алгоритм решения задачи на языке С++, обозначив массив ***a*** через `a`, элементы которого имеют тип **`int`**. Переменные *i*, *j*, *n*, *m*, *x* обозначим соответственно как `i, j, n, m, x` (все имеют тип **`int`**).

3. Текст программы

```
//программа перестановки элементов массива a[n]
#include <iostream>
#include <cmath>
using namespace std;
```

```
int main(){
   int i,j,m,x,n=18;
   int a[18];
   cout<<"Вводим массив a["<<n<<"]"<<endl;
   a[0]=40;
   for(i=0;i<=n-2; i++)a[i+1] =(37*a[i]+3)%64;
   for(i=0; i<=n-1; i++)cout<<a[i]<<" ";
   cout<<endl;
   m=n/2; j=n-1;
   for(i=0; i<=m-1; i++){
   x=a[i]; a[i]=a[j]; a[j]=x; j-=1;
   };
   cout<<"Массив a["<<n<<"] после
перестановки"<<endl;
   for(i=0; i<=n-1; i++)cout<<a[i]<<" ";
   cout<<endl;
}
```

4. Результаты работы программы

```
Вводим массив a[18]
40 11 26 5 60 47 14 9 16 19 2 13 36 55 54 17 56 27
Массив a[18] после перестановки
27 56 17 54 55 36 13 2 19 16 9 14 47 60 5 26 11 40
```

Контрольные вопросы

1. Какой тип могут иметь элементы массива?
2. Как происходит доступ к элементам массива?
3. Какой индекс имеют наименьший и наибольший элементы массива?
4. Как размещаются в памяти элементы массива?
5. Как растут индексы многомерных массивов?
6. Как присвоить значение элементов одного массива другому?

РАБОТА № 5

Разработка и реализация программы с использованием функций

Цель работы: овладение навыками составления программы с использованием функций и выполнения ее в NetBeans IDE.

Задания

Задание 5.1. Вычислить значения переменных, которые будут выведены на дисплей, после выполнения фрагмента программы по вариантам, приведенным в таблице 5.1. Указание: `n` равно номеру варианта.

Таблица 5.1 – Варианты задания 5.1

№	Фрагмент программы	№	Фрагмент программы

1–5	`void d(int &x, int &y);`
	`void main(){`
	`  int x,y,n;`
	`  cin>>n; x=3; y=4;`
	`  d(y,x); y=n*x;`
	`  cout<<x<<" "<<y;`
	`}`
	`void d(int &x, int &y){`
	`  x*=2; y=x+2;`
	`}`
		6–10	`int d(int x, int y);`
			`void main(){`
			`  int x,y,n;`
			`  cin>>n; x=3; y=4;`
			`  y=n*x+d(y,x);`
			`  cout<<x<<" "<<y;`
			`}`
			`int d(int x, int y){`
			`  return 2*x+y;`
			`}`

Продолж. табл. 5.1

№	Фрагмент программы	№	Фрагмент программы

11–15	`int d(int x, int y);`
	`void main(){`
	`  int x,y,n;`
	`  cin>>n; x=3; y=4;`
	`  y=n*x+d(n,y);`
	`  cout<<x<<" "<<y;`
	`}`
	`int d(int x, int y){`
	`  return 2*x-y;`
	`}`
		16–20	`void d(int &x, int &y);`
			`void main(){`
			`  int x,y,n;`
			`  cin>>n; n-=9; x=3; y=4;`
			`  d(y,x); y=n*x+y;`
			`  cout<<x<<" "<<y;`
			`}`
			`void d(int &x, int &y){`
			`  x*=2; y=x+1;`
			`}`

21–25	`int fp(int x);`
	`void main(){`
	`  int n,y; cin>>n; n-=18;`
	`  y=fp(n); cout<<y;`
	`}`
	`int fp(int x){`
	`  if(x>2)return x*fp(x-1);`
	`  else return x;`
		26–30	`int fs(int x);`
			`void main(){`
			`  int n,y; cin>>n; n-=21;`
			`  y=fs(n); cout<<y;`
			`}`
			`int fs(int x){`
			`  if(x>1)return x+fs(x-1);`
			`  else return x;`
			`}`

Задание 5.2. Составить программу вычисления величин из задания 4.2 с использованием функций и выполнить ее в NetBeans IDE.

Краткие теоретические сведения

Функция – специальная конструкция, с помощью которой фрагмент кода, который повторяется в программе несколько раз, выносится за тело основной программы.

Существует два способа объявления функции: до функции **main**; с помощью прототипа (тело объявленной функции описывается после функции **main**). При втором способе необходимо до функции **main** указать тип возвращаемого значения, имя функции и ее аргументы.

Общий синтаксис объявления функции:

тип_значения имя_функции (параметры) {

тело функции;

}

Возвращаемое значение – результат работы функции может быть любым из базовых или пользовательских типов. Если функция ничего не возвращает, на место возвращаемого значения, подставляется **void** (пусто).

Параметры (аргументы) – входные данные, необходимые для работы функции, для каждого параметра указывают тип данных. Параметры могут отсутствовать.

Параметры, которые указываются при определении функции, называются формальными, они создаются в момент вызова функции в оперативной памяти. При выходе из функции такие параметры будут уничтожены. Параметры, которые передаются в функцию при ее вызове, называются фактическими.

Формальному параметру функции может быть задано значение по умолчанию, параметрами по умолчанию могут быть параметры, начиная с правого конца списка без перерывов:

тип_значения имя_функции (тип_параметра имя_параметра = значение_по_умолчанию);

Нельзя создавать одну функцию внутри другой и вызвать функцию до ее объявления.

Для возвращения значения из функции в программу используется оператор **return**:

```
return res;
```

Если функция не возвращает никаких значений, оператор **return** можно использовать для остановки функции, в данном случае **return** отработает для функции, как **break** для цикла. Операторов **return** в функции может быть несколько, но сработает только один из них. Если тип значения, возвращаемого функцией, не **void**, то необходимо всегда использовать форму: **return** *значение;*

Вызов функции состоит из указания имени функции, передачи аргументов (при необходимости) и получения возвращаемого значения (при необходимости). При вызове функции необходимо указывать такое количество параметров, которое было определено при объявлении функции.

При использовании массива в качестве аргумента функции имя массива превращается в указатель на его первый элемент, то есть при передаче массива в качестве аргумента функции происходит передача указателя. При передаче одномерного массива достаточно указать пустые квадратные скобки:

```
int sum (int array[], int size);
```

При передаче двумерного массива необходимо указать количество столбцов, количество строк указывать не обязательно:

```
int sum (int array[][5], int size_row,
int size_col);
```

Любые операторные скобки в программном коде образуют так называемую область видимости. Переменные, объявленные внутри этих скобок, будет видно только в этой области видимости. Согласно правилам области видимости, переменные делятся на два вида – локальные и глобальные.

Локальные переменные создаются внутри любой области видимости. Глобальные переменные создаются вне всяких областей видимости, преимущественно в функции **main**(). Глобальная переменная видна в любом месте программы. По умолчанию глобальные переменные в отличие от локальных инициализируются 0. Те изменения, которые происходят с глобальной переменной внутри функции, при выходе из последней сохраняются.

Если глобальная и локальная переменные с одинаковыми именами, то внутри любой области видимости будет использоваться локальная переменная. Поэтому следует избе-

гать использования в программе одинаковых имен переменных.

Если функция вызывает сама себя, то она называется рекурсивной. Глубина рекурсии, то есть количество вызовов, в C++ не ограничивается. Реально она зависит от ресурсов памяти (размера стека).

В общем случае рекурсивность – это не свойство самой функции, а свойство ее описания. Рекурсивная функция, как правило, короче и нагляднее, чем нерекурсивная, но при выполнении требует больше времени и памяти за счет повторных обращений к самой себе и дублирования локальных переменных. Необходимо хорошо понимать, что каждый очередной рекурсивный вызов приводит к образованию новой копии локальных объектов функции и все эти копии, соответствующие цепочке активизированных и незавершенных рекурсивных вызовов, существуют независимо друг от друга и сохраняются в стеке. Это может привести к так называемому "взрыву" стека. "Взрыв" стека означает, что для записи локальных переменных функции не хватает памяти, отводимой под стек.

Пример выполнения работы

Задание 5.1. Вычислить значения переменных, которые будут выведены на дисплей, после выполнения следующего фрагмента программы:

```
   long fact(long x);
   void main(){
    long n,x,y; n=40; n-=36;
    y=fact(n); x=n; cout<<x<<" "<<y;
}
   long fact(long x){
    return x>2 ? x*fact(x-1):x;
   }
```

Решение

Эта программа вычисляет $x!$=4*3*2*1 с использованием рекурсивной функции. В результате ее выполнения x=4, а y=24.

Задание 5.2. Составить программу перестановки элементов массива ***a*** в обратном порядке с использованием функции и выполнить ее в NetBeans IDE. Элементы массива ***a*** определяются по формуле $a_{i+1}=(37\cdot a_i+3) \bmod 64$. Значение $a_0=40$; i изменяется от 0 до 16.

Решение

1. Постановка задачи

Составить программу перестановки элементов массива ***a*** в обратном порядке на языке C++ с использованием функции, если элементы массива определяются по формуле $a_{i+1}=(37\cdot a_i+3) \bmod 64$. Значение $a_0=40$; i изменяется от 0 до 16. Выполнить программу в NetBeans IDE.

2. Алгоритм решения задачи

Алгоритм приведен в примере решения задания 4.2 (при оформлении работы алгоритм необходимо привести заново).

3. Текст программы

```
//программа перестановки элементов массива a[n]
#include <iostream>
using namespace std;
void rev(int n, int a[]);
void main(){
   int i,j,m,x,n=18;
   int a[18];
   cout<<"Вводим массив a["<<n<<"]"<<endl;
   a[0]=40;
   for(i=0;i<=n-2; i++)a[i+1]= (37*a[i]+3)%64;
   for(i=0; i<=n-1; i++)cout<<a[i]<<" ";
```

```
   cout<<endl;
   rev(n, a);
   cout<<"Массив  a["<<n<<"]  после
перестановки"<<endl;
   for(i=0; i<=n-1; i++)cout<<a[i]<<" ";
   cout<<endl;
}
//функция перестановки элементов массива int a[n]
void rev(int n, int a[]){
   int m,i,j,x;
   m=n/2; j=n-1;
   for(i=0; i<=m-1; i++){
     x=a[i]; a[i]=a[j]; a[j]=x; j-=1;
   }
}
```

4. Результаты работы программы

```
Вводим массив a[18]
40 11 26 5 60 47 14 9 16 19 2 13 36 55 54 17 56 27
Массив a[18] после перестановки
27 56 17 54 55 36 13 2 19 16 9 14 47 60 5 26 11 40
```

Контрольные вопросы

1. Какие существуют способы объявления функции?
2. Какая разница между фактическими и формальными параметрами?
3. Как работает оператор **`return`**?
4. Что такое глобальная и локальная переменные?
5. Как осуществляется вызов функции?
6. Какая функция называется рекурсивной и какую особенность она имеет?

РАБОТА № 6

Разработка и реализация программы с использованием строк

Цель работы: овладение навыками составления программы с использованием строк и выполнения ее в NetBeans IDE.

Задания

Задание 6.1. Определить действие фрагмента программы по вариантам, которые приведены в таблице 6.1, если `char str1[20]="C++ language"; char *str2="12345"; int n;` (Указание: `n` равно номеру варианта).

Таблица 6.1 – Варианты задания 6.1

№	Фрагмент программы	№	Фрагмент программы
1–5	`strcat(str1,&str2[n-1]);` `cout<<str1;`	6–10	`strupr(str1);` `cout<<str1<<str2[n%5];`
11–15	`strlwr(str1);` `cout<<str1<<str2[n%11];`	16–20	`strcpy(str1,str2);` `cout<<str1<<str2[n%5];`
21–25	`strncpy(str1,str2,n%20);` `cout<<str1;`	26–30	`cout<<atoi(str2)/n;`

Задание 6.2. Составить программу нахождения следующих величин по вариантам, которые приведены в таблице 6.2, и выполнить ее в NetBeans IDE, если дана строка "We study C++ programming language first semester.".

Таблица 6.2 – Варианты задания 6.2

№	Величины, которые необходимо найти
1–3	Количество слов в строке
4–6	Количество букв е в строке
7–9	Длину самого короткого слова в строке
10–12	Длину самого длинного слова в строке
13–15	Количество заглавных букв в строке
16–18	Первые буквы каждого слова
19–21	Слова, которые не содержат букву е
22–24	Какие буквы и сколько раз встречаются в строке
25–27	Каких букв больше в строке: гласных или согласных
28–30	Позиции символов, которые не являются буквами

Краткие теоретические сведения

В языке С++, как и в языке С, нет встроенной поддержки срочного типа данных, для работы со строками используются символьные массивы, в каждом элементе хранится один символ и занимает один байт, потому что каждый элемент символьного массива имеет тип **`char`**. Признаком конца строки является нуль-терминатор `\0` (нулевой символ). Нуль-символ – это не цифра 0, в таблице кодов ASCII он имеет номер 0 и не выводится на печать. Наличие нуль-символа означает, что количество элементов массива должно быть хотя бы на один больше, чем количество символов в строке. При использовании в выражениях строка заключается в двойные кавычки. Кавычки не является частью строки, они отмечают ее начало и конец.

Работа со строками с использованием класса `string` будет описана в части 2 Методических указаний.

Варианты инициализации строк:

– объявить массив типа **`char`**, с помощью оператора присваивания = в двойных кавычках указать необходимый текст, нуль-терминатор `\0` добавится автоматически, размер массива при этом указывать не обязательно (определяется компилятором): **`char`** `str[]="Hello world";` Используя пустые кавычки при инициализации, мы присваиваем каждому элементу

массива значение \0. Таким образом строка будет очищена от "мусора" других программ.

– объявить массив типа **char**, текст не присваивать, при этом указать размер массива: **char** str[20];

Классический способ объявления массива **char** str[100] = {'H','e','l','l','o',' ','w', 'o','r','l','d','\0'}; при работе со строками не используется.

Основные функции для работы со строками приведены в приложении Е. Для вывода строки на экран достаточно обратиться к ней по имени: cout<<str<<endl; cout будет выводить на экран символ за символом, пока не встретит в одном из элементов массива символ конца строки \0 и вывод прервется. Такое обращение для обычного символьного массива (массива без нуль-терминатора \0) недопустимо. Также для вывода строки на экран используется функция puts().

Ввод строки с клавиатуры можно осуществить с помощью команды cin, однако из всего введенного считается последовательность символов до первого пробела. Более универсальной является функция gets(), которая считывает все введенные символы с пробелами, пока не будет нажата клавиша Enter.

Также для работы со строками используются указатели. В этом случае к каждой строке можно обратиться за помощью указателя на ее первый символ. Более подробно работа с указателями будет описана в части 2 Методических указаний.

Пример выполнения работы

Задание 6.1. Определить действие фрагмента программы

```
char  *str="12345";
cout<<atof(str)/5;
```

Решение

Этот фрагмент программы выводит на экран число 2469.

Задание 6.2. Составить программу, которая удаляет все пробелы из строки и выполнить ее в NetBeans IDE, если дана строка "We study C++.".

Решение

1. Постановка задачи

Составить программу, которая удаляет все пробелы из строки "We study C++.". Выполнить программу в NetBeans IDE.

2. Алгоритм решения задачи

Алгоритм решения задачи можно представить в виде такой последовательности действий:

Действие 1. Инициализировать строку str.

Действие 2. Вывести строку str.

Действие 3. Определить *m* (длину строки str).

Действие 4. Повторять *m* раз следующие действия (*i*= 0, 1, ..., *m*-1):

Действие 4.1. *j*-му символу строки str присвоить значение *i*-го символа строки str;

Действие 4.2. Если *i*-й символ строки str не пробел, то увеличить значение счетчика *j* на 1;

Действие 4.3. Увеличить значение счетчика *i* на 1.

Действие 5. Если *j*<*i*, то *j*-му символу строки str присвоить значение конца строки `\0`.

Действие 6. Вывести строку str после преобразования.

Запишем алгоритм решения задачи на языке C++, обозначив строку через `str`, символы которой имеют тип **`char`**. Переменные *i* и *m* обозначим соответственно как `i`, `m`, которые имеют тип **`int`**.

3. Текст программы

```
//программа удаление пробелов из строки str
#include <iostream>
#include <cstring>
using namespace std;
int main() {
```

```
   char str[]="We study C++.";
   int m, i=0, j=0;
   cout<<"Строка: "<<str<<endl;
   m=strlen(str);
   while (i<m) {
     str[j]=str[i];
     if (str[i]!=' ') j+=1;
     i+=1;
   }
   if (j<i) str[j] = '\0';
   cout<<"Строка после преобразования:
"<<str<<endl;
}
```

4. Результаты работы программы

```
Строка: We study C++.
Строка после преобразования: WestudyC++.
```

Контрольные вопросы

1. Что используется для работы со строками?
2. Что значит нуль-терминатор \0?
3. Какие существуют способы инициализации строк?
4. Как можно вывести строку на экран?
5. Как можно ввести строку с клавиатуры?
6. Приведите основные функции работы со строками.

СПИСОК РЕКОМЕНДОВАННОЙ ЛИТЕРАТУРЫ

1. **Буч, Г.** Объектно-ориентированное проектирование с примерами применения [Текст] : пер. с англ. / Г. Буч. – М. : Конкорд, 1992. – 519 с.

2. **Дьюхарт, С.** Программирование на Си++ [Текст] / С. Дьюхарт, К. Старк. – К. : НИПФ "ДиаСофт", 1993. – 272 с.

3. **Страуструп, Б.** Программирование на Си++ [Текст] : пер. с англ. / Б. Страуструп. – М. : Радио и связь, 1991.– 352 с.

4. **Страуструп, Б.** Язык программирования С++ [Текст] / Б. Страуструп. – М. : Бином, 2004.– 369 с.

5. **Страуструп, Б.** Язык программирования С++. Специальное издание [Текст] / Б. Страуструп. – М. : Бином, 2011.– 1136 с.

6. **Шилдт, Г.** Самоучитель С++. [Текст] : 3-е издание [Текст] : пер. с англ. / Г. Шилдт. – СПб. : BHV – Санкт-Петербург, 1998. – 688 с.

7. **Шилдт, Г.** С++: базовый курс. [Текст] : 3-е издание [Текст] : пер. с англ. / Г. Шилдт. – М. : Издательский дом "Вильямс", 2010. – 624 с.

8. Borland C++ [Текст] : пер. с англ. / П. Киммел [и др.]. – СПб. : BHV – Санкт-Петербург, 1997. – 976 с.

Приложение А
Алфавит языка С++

Алфавит языка С++ составляют:

– буквы латинского алфавита: `A – Z, a – z` и символ подчеркивания _ (ASCII-код 95);

– арабские цифры: `0 – 9`;

– специальные символы: `+ – * / % = < > . , : ; ' " ( ) [ ] { } # $ ^ ~ ! ? & |`;

– символы-разделители: символ пропуска (ASCII-код 32) и управляющие символы (приведены в приложении Б);

– служебные (зарезервированные) слова (приведены в приложении В).

Приложение Б
Управляющие символы языка C++

Последовательности символов, начинающиеся с обратной косой черты (\), называют управляющими, или escape-последовательностями – символы, которые выталкиваются в поток вывода с целью форматирования вывода или печати некоторых управляющих знаков C++. Основной список управляющих символов C++ представлен в таблице Б.1.

Управляющие последовательности используются для описания определенных специальных символов внутри строковых литералов и рассматриваются как один символ.

Таблица Б.1 – Управляющие символы языка C++

Управляющая последовательность	Описание
`\'`	одинарная кавычка
`\"`	двойная кавычка
`\?`	литерал знака вопроса
`\\`	обратный слеш
`\0`	нулевой символ
`\a`	звуковой сигнал
`\b`	удаление предыдущего символа
`\f`	новая страница
`\n`	новая строка
`\r`	возврат каретки
`\t`	горизонтальная табуляция
`\v`	вертикальна табуляция
`\nnn`	символ ASCII в восьмеричной нотации
`\xnn`	символ ASCII в шестнадцатеричной нотации
`\unnnn`	символ юникода в шестнадцатеричной нотации, если эта escape-последовательность используется в многобайтовой знаковой константе или строчном литерале юникода

Приложение В
Служебные слова языка C++

Служебные (зарезервированные) слова – это ограниченная группа слов, смысл которых зафиксирован в языке. Служебные слова нельзя употреблять как идентификаторы переменных, констант и т. п.

```
asm           auto         bool              break
case          catch        char              class
const         const_cast   continue          default
delete        do           double            dynamic_cast
else          enum         explicit          extern
false         float        for               friend
goto          if           inline            int
long          mutable      namespace         new
operator      overload     private           protected
public        register     reinterpret_cast  return
short         signed       sizeof            static
static_cast   struct       switch            template
this          throw        true              try
typedef       typeid       typename          union
unsigned      using        virtual           void
volatile      wchar_t      while
```

Идентификаторы `signed` и `volatile` зарезервированы для использования в будущем.

Приложение Г
Математические функции языка С++

Базовые математические функции языка программирования С++ содержатся в заголовочном файл cmath.h (аналог файла math.h для языка С). Первоначально в языках С и С++ поддерживался один и тот же набор из 22 математических функций. По мере развития языка С++ к нему добавились перегруженные версии этих оригинальных функций, которые явно предназначены для приема значений типа float и long double (в отличие от типа double в языке С). Действия, выполняемые функциями, остались теми же. Все углы задаются в радианах. В таблице Г.1 приведены базовые математические функции языка С++.

Таблица Г.1 – Базовые математические функции языка С++

Функция	Описание
acos(a)	арккосинус a, где -1.0 < a < 1.0
asin(a)	арксинус a, где -1.0 < a < 1.0
atan(a)	арктангенс a
atan2(a, b)	арктангенс b/a
ceil(a)	наименьшее целое, которое превышает или равно a
cos(a)	косинус a
cosh(a)	гиперболический косинус a
exp(a)	значение экспоненты (e^a)
abs(a)	абсолютное значение (модуль) a
floor(a)	наибольшее целое, которое меньше или равно a
fmod(a, b)	остаток от деления a/b
frexp(a, int *exp)	разбивает число a на мантиссу и экспоненту
ldexp(a, int exp)	возвращает значение выражения $a*2^{exp}$
log(a)	натуральный логарифм a
log10(a)	десятичный логарифм a
modf(a, *i)	разбивает a на целую и дробную части
pow(a, b)	возведение a в степень b
sin(a)	синус a
sinh(a)	гиперболический синус a
sqrt(a)	корень квадратный из a, где a больше или равно 0
tan(a)	тангенс a
tanh(a)	гиперболический тангенс a

Приложение Д
Операции языка C++ по приоритету

В таблице Д.1 приведены операции по приоритету с описанием их обозначений, названий и синтаксиса. Самый высокий приоритет имеют операции в верхней части таблицы.

Таблица Д.1 – Операции языка C++ по приоритету

Обозначение	Название	Синтаксис
`::`	Разрешение области видимости	*имя_класса* `::` *элемент*
`::`	Глобальное имя	`::` *имя*
`.`	Выбор элемента через объект	*объект* `.` *элемент*
`->`	Выбор элемента через указатель	*указатель* `->` *элемент*
`[]`	Элемент массива	*указатель* `[`*выражение*`]`
`()`	Вызов функции	*выражение* `(`*аргументы*`)`
`()`	Задание значения	*тип* `(`*аргументы*`)`
`sizeof`	Размер объекта	`sizeof` *выражение*
`sizeof`	Размер типа	`sizeof (`*тип*`)`
`++`	Постфиксный инкремент	*идентификатор*`++`
`++`	Префиксный инкремент	`++`*идентификатор*
`--`	Постфиксный декремент	*идентификатор*`--`
`--`	Префиксный декремент	`--`*идентификатор*
`~`	Дополнение	`~`*выражение*
`!`	Отрицание	`!`*выражение*
`-`	Унарный минус	`-`*выражение*
`+`	Унарный плюс	`+`*выражение*
`&`	Получение адреса	`&`*идентификатор*
`*`	Разыменование	`*`*выражение*
`new`	Создание	`new` *тип*
`new[]`	Создание массива	`new` *тип*`[]`
`delete`	Удаление	`delete` *указатель*
`delete[]`	Удаление массива	`delete[]`*указатель*
`()`	Приведение типа	`(`*тип*`)` *выражение*
`.*`	Выбор указателя через объект	*объект* `.*`*указатель-элемента*
`->*`	Выбор указателя через указатель	*указатель*`->*` *указатель-элемента*

Продолж. табл. Д.1

Обозначение	Название	Синтаксис
*	Умножение	*выражение* * *выражение*
/	Деление	*выражение* / *выражение*
%	Остаток от деления	*выражение* % *выражение*
+	Сложение	*выражение* + *выражение*
-	Вычитание	*выражение* - *выражение*
<<	Сдвиг влево	*выражение* << *выражение*
>>	Сдвиг вправо	*выражение* >> *выражение*
<	Меньше	*выражение* < *выражение*
<=	Меньше или равно	*выражение* <= *выражение*
>	Больше	*выражение* > *выражение*
>=	Больше или равно	*выражение* >= *выражение*
==	Равно	*выражение* == *выражение*
!=	Не равно	*выражение* != *выражение*
&	Побитовое И	*выражение* & *выражение*
^	Побитовое исключающее ИЛИ	*выражение* ^ *выражение*
|	Побитовое ИЛИ	*выражение* | *выражение*
&&	Логическое И	*выражение* && *выражение*
||	Логическое ИЛИ	*выражение* || *выражение*
?:	Условное выражение	*выражение*? *выражение*: *выражение*
=	Простое присваивание	*идентификатор* = *выражение*
*=	Умножение и присваивание	*идентификатор* *= *выражение*
/=	Деление и присваивание	*идентификатор* /= *выражение*
%=	Остаток от деления и присваивание	*идентификатор* %= *выражение*
+=	Сложение и присваивание	*идентификатор* += *выражение*
-=	Вычитание и присваивание	*идентификатор* -= *выражение*
<<=	Сдвиг влево и присваивание	*идентификатор* <<= *выражение*
>>=	Сдвиг вправо и присваивание	*идентификатор* >>= *выражение*
&=	Побитовое И и присваивание	*идентификатор* &= *выражение*
|=	Побитовое ИЛИ и присваивание	*идентификатор* |= *выражение*
^=	Побитовое исключающее ИЛИ и присваивание	*идентификатор* ^= *выражение*

Приложение Е
Функции для работы со строками языка С++

Основные функции для работы со строками языка программирования С++ содержатся в нескольких заголовочный файлах: cstring (аналог файла string.h для языка С), приведены в таблице Е.1; cstdio (аналог файла stdio.h для языка С), приведены в таблице Е.2; cstdlib (аналог файла stdlib.h для языка С), приведены в таблице Е.3.

Таблица Е.1 – Основные функции для работы со строками файла cstring

Функция	Описание
`int strlen(char* str)`	подсчитывает длину строки (количество символов без учета \0)
`char *strcat(char *dest, const char *scr)`	присоединяет строку src в конец строки dest, полученная строка возвращается в качестве результата
`char *strncat(char *dest, const char *scr, int n)`	присоединяет n символов строки src в конец строки dest и возвращает строку dest
`char *strcpy(char *dest, const char *scr)`	выполняет копирование строки src в строку dest и возвращает строку dest
`char *strncpy(char *dest, const char *scr, int n)`	выполняет копирование n символов строки src в строку dest и возвращает строку dest
`int ctrcmp(const char *s1, const char *s2)`	сравнивает две строки в лексикографическом порядке с учетом различия заглавных и строчных букв; функция возвращает 0, если строки совпадают, возвращает -1, если s1 располагается в упорядоченном по алфавиту порядке ранее, чем s2, и 1 – в противном случае
`int ctrncmp(const char *s1, const char *s2, int n)`	сравнивает n символов двух строк в лексикографическом порядке с учетом различия заглавных и строчных букв; функция возвращает 0, если строки совпадают, возвращает -1, если s1 располагается в упорядоченном по алфавиту порядке ранее, чем s2, и 1 – в противном случае

Продолж. табл. Е1

Функция	Описание
`getline(char *s, int n)`	предназначена для ввода с клавиатуры строки s с пробелами, в строке не должно быть больше n символов
`char *strchr(const char *str, int c);`	возвращает указатель на первое вхождение символа с в строку, на которую указывает str, если символ с не найден, возвращает NULL
`char *strupr(char *str)`	превращает символы строки, на которую указывает str, в символы верхнего регистра, после чего возвращает ее
`char *strlwr(char *str)`	превращает символы строки, на которую указывает str, в символы нижнего регистра, после чего возвращает ее

Таблица Е.2 – Основные функции для работы со строками файла cstdio

Функция	Описание
`int sprintf(char*, const char*, ...)`	вывод результата в строку
`int sscanf(const char*, const char*, ...)`	ввод значений из строки
`char* gets(char*)`	считывает поток символов из стандартного устройства ввода в строку до тех пор, пока не будет нажата клавиша ENTER
`int puts(const char*)`	вывод строки на экран с переводом курсора на следующую строку

Таблица Е.3 – Функции преобразования для работы со строками файла cstdlib

Функция	Описание
`int atoi(const char *str)`	преобразует строку в целое число, в случае неудачного преобразования возвращается 0
`long atol(const char *str)`	преобразует строку в длинное целое число, в случае неудачного преобразования возвращается 0
`double atof(const char *str)`	преобразует строку в действительное число, в случае неудачного преобразования возвращается 0

Приложение Ж
Использование манипуляторов в языке C++

Система ввода/вывода языка C++ включает способ изменения параметров форматирования потока. Для этого используются специальные функции, называемые манипуляторами (manipulators), которые могут включаться в выражения ввода/вывода. Описание стандартных манипуляторов находится в заголовочном файле iomanip.h и приведено в таблице Ж.1. Для использования манипуляторов с параметрами в программу необходимо включить заголовочный файл iomanip.h.

Таблица Ж.1 – **Стандартные манипуляторы языка C++**

Манипулятор	Описание	Использование
`dec`	Ввод/вывод данных в десятичной форме	ввод и вывод
`endl`	Вывод символа новой строки с передачей в поток всех данных из буфера	вывод
`ends`	Вывод нулевого символа	вывод
`flush`	Передача в поток содержимого буфера	вывод
`hex`	Ввод/вывод данных в шестнадцатеричной системе	ввод и вывод
`oct`	Ввод/вывод данных в восьмеричной форме	ввод и вывод
`resetiosflags(long f)`	Сбрасывает флаги, указанные в f	ввод и вывод
`setbase(int base)`	Устанавливает базу счисления равной параметру base	вывод
`setfill(int ch)`	Устанавливает символ заполнения равным ch	вывод
`setiosflags(long f)`	Устанавливает флаги, указанные в f	ввод и вывод
`setprecision(int p)`	Устанавливает число цифр после запятой	вывод
`setw(int w)`	Устанавливает ширину поля равной w	вывод
`ws`	Пропускает начальный символ-разделитель	ввод

Манипуляторы могут использоваться в составе выражений ввода/вывода. Ниже представлен пример программы, использующей манипуляторы для изменения формата вывода. Следует обратить внимание, как манипуляторы появляются в последовательности операторов ввода/вывода. Когда манипуляторы не имеют аргументов, как, например, манипулятор `endl`, за ними не следуют скобки. Причина этого в том, что оператору << передается адрес манипулятора.

```
#include <iostream>
#include <iomanip>
using namespace std;
int main()
{
cout<<fixed;
cout<<setprecision (5)<<45.25643358<<endl;
cout<<setfill(':')<<setw(20)<<"Hello.";
return 0;
}
```

Результат работы программы:

```
45.25643
::::::::::::::Hello.
```

Приложение 3
Оценка погрешности при работе с действительными числами на цифровом компьютере

Погрешность при работе с действительными числами на цифровом компьютере может состоять из трех составляющих:

– погрешность представления;

– погрешность накопления;

– погрешность используемого численного метода.

Необходимо иметь возможность оценивать эту погрешность, т.е. оценивать с какой точностью будет представлен результат, а также уменьшать ее.

Для уменьшения погрешности представления необходимо использовать вещественные типы данных с большим числом байт, а для уменьшения погрешности накопления – специальные приемы программирования, которые ее минимизируют.

Проиллюстрируем данные составляющие погрешности на примере табулирования функции $y=x^3$ при изменении значения x от $a=-1$ до $b=1$ с шагом $h=0{,}1$.

Пример 1.

```
#include <iostream>
#include <cmath>
#include <iomanip>
using namespace std;
int main(){
float a=-1,b=1,h=0.1,x;
double mas[]={-1.0,-0.9,-0.8,-0.7,-0.6,-0.5,-0.4,-0.3,
20.1,0.0,0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9,1.0};

cout<<fixed<<setprecision(10)<<endl;
cout<<"h="<<h<<endl;
cout<<"----------"<<endl;
int j;
for(x=a, j=0; x<=b;x+=h,j++)
cout<<"x="<<x<<"  y="<<pow(x,3)<<"  delta
y="<<abs(pow(x,3)-pow(mas[j],3))<<endl;
```

```
cout<<"----------"<<endl;
x+=h;
cout<<"x="<<x<<"   y="<<pow(x,3)<<"   delta
y="<<abs(pow(x,3)-pow(mas[j],3))<<endl;
return 0;
}
```

Результат работы программы:

```
h=0.1000000015
----------
x=-1.0000000000 y=-1.0000000000 delta y=0.0000000000
x=-0.8999999762 y=-0.7289999127 delta y=0.0000000873
x=-0.7999999523 y=-0.5119999051 delta y=0.0000000949
x=-0.6999999285 y=-0.3429998755 delta y=0.0000001245
x=-0.5999999046 y=-0.2159999013 delta y=0.0000000987
x=-0.4999999106 y=-0.1249999329 delta y=0.0000000671
x=-0.3999999166 y=-0.0639999583 delta y=0.0000000417
x=-0.2999999225 y=-0.0269999783 delta y=0.0000000217
x=-0.1999999285 y=-0.0079999920 delta y=0.0000000080
x=-0.0999999270 y=-0.0009999978 delta y=0.0000000022
x=0.0000000745 y=0.0000000000 delta y=0.0000000000
x=0.1000000760 y=0.0010000024 delta y=0.0000000024
x=0.2000000775 y=0.0080000097 delta y=0.0000000097
x=0.3000000715 y=0.0270000193 delta y=0.0000000193
x=0.4000000656 y=0.0640000328 delta y=0.0000000328
x=0.5000000596 y=0.1250000447 delta y=0.0000000447
x=0.6000000834 y=0.2160000950 delta y=0.0000000950
x=0.7000001073 y=0.3430001736 delta y=0.0000001736
x=0.8000001311 y=0.5120002627 delta y=0.0000002627
x=0.9000001550 y=0.7290003896 delta y=0.0000003896
----------
x=1.1000001431 y=1.3310004473 delta y=0.3310004473
```

В приведенном примере вначале программы (до цикла) иллюстрируется погрешность представления: `float h=0.1` при форматном выводе на экран `setprecision(10)` дает `h=0.1000000015`. В данном случае погрешность представления составляет `0.0000000015` или $1.5*10^{-9}$.

В примере 1 последующее значение переменной x в цикле вычисляется как $x_i=x_{i-1}+h$, что приводит к увеличению погрешности накопления с каждой итерацией цикла. Кроме того, из-за полученной погрешности накопления значение функции y при x=1 в цикле получить не удалось (это значение посчитано отдельно после цикла).

Погрешность накопления возможно уменьшить, если значение переменной x в цикле вычислять по следующей формуле $x_i=a+i*h$, что показано в примере 2.

Пример 2.

```
#include <iostream>
#include <cmath>
#include <iomanip>
using namespace std;
int main(){
double a=-1,b=1,h=0.1,x;
double  mas[]={-1.0,-0.9,-0.8,-0.7,-0.6,-0.5,
-0.4,-0.3,-0.2,0.1,0.0,0.1,0.2,0.3,0.4,0.5,
0.6,0.7,0.8,0.9,1.0};
cout<<fixed<<setprecision(18)<< endl;
cout<<"h="<<h<<endl;
cout<<"----------"<<endl;
int i=0, j;
for(x=a, j=0; x<=b;x=a+i*h,j++){
cout<<"x="<<x<<"   y="<<pow(x,3)<<"   delta
y="<<abs(pow(x,3)-pow(mas[j],3))<<endl;
i++;
}
return 0;
}
```

Результат работы программы:

```
h=0.100000000000000010
----------
x=-1.000000000000000000 y=-1.000000000000000000 delta
y=0.000000000000000000
x=-0.900000000000000020 y=-0.729000000000000090 delta
y=0.000000000000000000
```

x=-0.800000000000000040 y=-0.512000000000000120 delta
y=0.000000000000000000
x=-0.699999999999999960 y=-0.342999999999999920 delta
y=0.000000000000000000
x=-0.599999999999999980 y=-0.216000000000000000 delta
y=0.000000000000000000
x=-0.500000000000000000 y=-0.125000000000000000 delta
y=0.000000000000000000
x=-0.399999999999999970 y=-0.063999999999999987 delta
y=0.000000000000000028
x=-0.299999999999999930 y=-0.026999999999999979 delta
y=0.000000000000000021
x=-0.199999999999999960 y=-0.007999999999999995 delta
y=0.000000000000000007
x=-0.099999999999999950 y=-0.000999999999999999 delta
y=0.000000000000000002
x=0.000000000000000056 y=0.000000000000000000 delta
y=0.000000000000000000
x=0.100000000000000060 y=0.001000000000000002 delta
y=0.000000000000000002
x=0.200000000000000070 y=0.008000000000000009 delta
y=0.000000000000000007
x=0.300000000000000040 y=0.027000000000000010 delta
y=0.000000000000000010
x=0.400000000000000080 y=0.064000000000000029 delta
y=0.000000000000000014
x=0.500000000000000110 y=0.125000000000000080 delta
y=0.000000000000000083
x=0.600000000000000090 y=0.216000000000000080 delta
y=0.000000000000000083
x=0.700000000000000070 y=0.343000000000000080 delta
y=0.000000000000000167
x=0.800000000000000040 y=0.512000000000000120 delta
y=0.000000000000000000
x=0.900000000000000130 y=0.729000000000000310 delta
y=0.000000000000000222
x=1.000000000000000000 y=1.000000000000000000 delta
y=0.000000000000000000

В начале примера 2 за счет использования типа данных **double** (размер – 8 байт) вместо **float** (размер – 4 байта) удалось уменьшить погрешность представления: `double h=0.1` при форматном выводе на экран `setprecision(18)` дает `h=0.10000000000000010`. В данном случае погрешность представления составляет `0.00000000000000010` или $1.0*10^{-17}$.

Как видно из результатов работы программы (пример 2), удалось также уменьшить погрешность накопления. В третьем столбце приведено значение абсолютной погрешности $\Delta y=|y_{\text{точное}}-y_{\text{приближенное}}|$. В примере 1 Δy (`delta y`) увеличивается и существенно больше, чем в примере 2.

Оценка и уменьшение погрешности ряда численных методов будет рассмотрена в дисциплине "Численные методы".

СОДЕРЖАНИЕ