МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет кораблебудування
імені адмірала Макарова

**С. Б. ПРИХОДЬКО, Л. М. МАКАРОВА,
Т. Г. СМИКОДУБ**

МЕТОДИЧНІ ВКАЗІВКИ
до виконання лабораторних робіт з дисципліни «Основи програмування»

У двох частинах

Частина 1

Рекомендовано Методичною радою НУК

ВИДАВНИЦТВО
НАЦІОНАЛЬНОГО УНІВЕРСИТЕТУ
КОРАБЛЕБУДУВАННЯ
ІМ. АДМІРАЛА МАКАРОВА
2022

УДК 004.4(076)
П 77

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

Рецензент М. В. Ушкац, д-р фіз.-мат. 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, 2022. – 60 с.

Дані методичні вказівки мають допомогти здобувачам у підготовці та виконанні лабораторних робіт з дисципліни «Основи програмування». Вони містять завдання для лабораторних робіт, стислий виклад теоретичних відомостей, етапи виконання робіт та контрольні запитання.

Призначено для здобувачів першого курсу спеціальності 121 «Інженерія програмного забезпечення», а також для всіх, хто вивчає основи програмування мовою С++.

УДК 004.4(076)

ЗМІСТ

ВСТУП

В Національному університеті кораблебудування імені адмірала Макарова (НУК) дисципліна «Основи програмування» вивчається здобувачами спеціальності 121 «Інженерія програмного забезпечення» у першому та другому семестрах. Вона відноситься до циклу загальної підготовки обов`язкових компонент освітньої програми.

Дані методичні вказівки мають допомогти здобувачам першого курсу спеціальності 121 «Інженерія програмного забезпечення» у підготовці та виконанні лабораторних робіт з дисципліни «Основи програмування». Вони містять завдання для лабораторних робіт, стислий виклад теоретичних відомостей, етапи виконання робіт та контрольні запитання.

При виконанні кожної лабораторної роботи рекомендується:

- засвоїти теоретичні відомості;
- розібратися з завданням та етапами виконання роботи;
- виконати запропоновані завдання та оформити звіт з роботи;
- перевірити повноту розуміння теми за допомогою контрольних запитань, розташованих у кінці кожної роботи.

Лабораторну роботу необхідно виконувати відповідно до наведених етапів виконання роботи. Звіт про лабораторну роботу оформлюється кожним здобувачем індивідуально. Звіт повинен містити: назву і мету роботи; завдання (постановку задачі); методику й алгоритм розв’язання задачі; текст програми; результати роботи; аналіз результатів та висновки.

Робота № 1
Розробка та реалізація програми з лінійною структурою

Мета роботи: закріплення знань алфавіту мови програмування С++, набуття навичок запису констант, змінних, виразів, операторів присвоєння; оволодіння навичками складання програми з лінійною структурою.

Завдання

Завдання 1.1. Записати мовою С++ математичні вирази за варіантами, які наведені в таблиці 1.1.

Таблиця 1.1 – Варіанти завдання 1.1

№	Математичні вирази	№	Математичні вирази
1-3	а) $3(z+1)^2+2,1\cdot10^6$ б) $\|x+z\|>0 \wedge 0<b<1$	4-6	а) $10^{-4}\, e^{-2\,f} +\|z^3\|$ б) $x+z<0 \vee 0<f<2$
7-9	а) $3(z+3)^{0,5}+10^{-3}$ б) $\|x+z\|>1 \wedge 1<b<2$	10-12	а) $\ln(z+1)^{0,5}+4,2\cdot10^4$ б) $\|x\|>2 \vee 0<b<3$
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) $\cos\|x+z\|>0 \vee 0<b<6$
19-21	а) $7(z+1)^{0,5}+2,7\cdot10^6$ б) $\|x+z\|>0 \wedge 0<b<7$	22-24	а) $10^{-7}\sin\|3z\| + b^{0,5}$ б) $\ln\|x+z\|>0 \vee 0<b<1$
25-27	а) $\|z+1\|^3+2\cdot10^5$ б) $\ln\|x+z\|>0 \wedge b>9$	28-30	а) $(z+1)^{0,5}+10^6$ б) $\|x+z\|>0 \vee 0<b< 1$

Прийняті позначення: $\vee$ – об’єднання множин, $\wedge$ – переріз множин.

Завдання 1.2. Представити математичний запис виразу за варіантами, які наведені в таблиці 1.2, і показати порядок дій.

Таблиця 1.2 – Варіанти завдання 1.2

№	Вираз
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.)`

Завдання 1.3. Скласти програму обчислення наступних величин за варіантами, які наведені в таблиці 1.3, та виконати її.

Таблиця 1.3 – Варіанти завдання 1.3

№	Величини, які треба обчислити
1	Модуль вектора $5\boldsymbol{a}+10\boldsymbol{b}$, якщо $\boldsymbol{a}=\{3; 2\}$ і $\boldsymbol{b}=\{0; -1\}$
2	Проекція вектора $\boldsymbol{a}=\{5; 2; 5\}$ на вісь вектора $\boldsymbol{b}=\{2; -1; 2\}$
3	Периметр трикутника з вершинами A(1; 1), B(4; 1), C(4; 5)
4	Модуль вектора $-2\boldsymbol{a}+4\boldsymbol{b}$, якщо $\boldsymbol{a}=\{3; 2\}$ і $\boldsymbol{b}=\{0; -1\}$
5	Кути трикутника з вершинами A(0; 1,7), B(2; 1,7), C(1,5; 0,85)
6	Площа чотирикутника з вершинами A(0;0), B(–1;3), C(2;4), D(3;1)
7	Модуль вектора $\boldsymbol{a}+\boldsymbol{b}$, якщо $\lvert\boldsymbol{a}\rvert=11$, $\lvert\boldsymbol{b}\rvert=23$, $\lvert\boldsymbol{a}-\boldsymbol{b}\rvert=30$
8	Модуль вектора $\boldsymbol{a}-\boldsymbol{b}$, якщо $\lvert\boldsymbol{a}\rvert=3$, $\lvert\boldsymbol{b}\rvert=5$, та $\boldsymbol{a}$ і $\boldsymbol{b}$ утворюють кут у 120^0
9	Кут між векторами $\boldsymbol{a}=\{2; -4; 4\}$ і $\boldsymbol{b}=\{-3; 2; 6\}$
10	Модуль вектора $\boldsymbol{a}+\boldsymbol{b}$, якщо $\boldsymbol{a}=\{1; 2; 3\}$ і $\boldsymbol{b}=\{4; -2; 9\}$

Короткі теоретичні відомості

Алфавіт мови C++ складається з сукупності символів, які наведені у додатку А. За допомогою символів алфавіту можна складати різноманітні конструкції: константи, змінні, оператори тощо. Управляючі символи мови C++, або escape-послідовності, наведені у додатку Б.

Постійна величина (константа) не змінюється в процесі роботи програми. Є декілька видів констант: числові (цілі та з плаваючою крапкою), символьні, логічні.

Цілі десяткові константи складаються з послідовності десяткових цифр, перед якою немає цифри `0`, а може стояти знак “+” або “–”. Приклад: `-15`.

Цілі вісімкові константи складаються з послідовності вісімкових цифр, перед якою є цифра `0` та може стояти знак “+” або “–”. Приклад: `-015`.

У мові C++ цілі константи можна зображати і в шістнадцятковій системі числення. В цьому випадку константи починаються з `0x` або `0X`, за якими йдуть шістнадцяткові цифри. Перед знаком `0x` або `0X` може бути знак “+” або “–”. Приклад: `0x2A`.

Цілі десяткова, вісімкова та шістнадцяткова константа, значення якої перевищує найбільше машинне ціле зі знаком, є довгою константою (`long`); в інших випадках цілі константи вважаються `int`.

Цілі десяткова, вісімкова та шістнадцяткова константа, за якою безпосередньо стоїть буква `l` або `L`, є довгою константою (`long`).

Константа з плаваючою крапкою складається з цілої частки, десяткової крапки, дробової частки, букви `e` або `E` та цілого показника степеня. Ціла або дробова частка (але не обидві одразу) може бути відсутньою. Десяткова точка, або `e` (`E`) сумісно з цілим показником степеня (але не обидві частки одночасно) може бути відсутньою. Наприклад, число 0,015 можна записати як `0.015` та `0.15E–01`, або `1.5E–02`. Константа з плаваючою крапкою має тип `double`.

Символьна константа – це символ у лапках. Приклад: `'f'`. Символьні константи вважаються даними типу `int`.

Логічні константи приймають тільки два значення: `true` (істина) та `false` (хибність).

Коментар – це текст, обмежений символами `/*` і `*/`, або текст, який починається символами `//` та закінчується наприкінці рядка. Коментарі `/* */` не можуть бути вкладеними. Символи `//` можна використовувати для того, щоб закоментувати символи `/*` або `*/`, а символами `/*` можна закоментувати `//`.

Змінна визначається ім’ям (ідентифікатором), типом та значенням.

Ідентифікатор є послідовністю букв і цифр, причому перший символ повинен бути буквою. Символ підкреслювання “_” вважається буквою. В ідентифікаторах великі та малі букви розрізнюються, букви кирилиці використовувати не можна. Не можна використовувати як ідентифікатори службові слова, список яких наведений у додатку В.

Мова C++ є суворо типізованою мовою. Це означає, що кожна змінна повинна бути описана, тобто необхідно визначити її тип.

Обробка даних виконується в виразах. Вираз складається з операцій і операндів. Операндом може бути константа, змінна або ім’я функції (математичні функції наведені у додатку Г). Обчислення виконуються

зліва направо з урахуванням пріоритету операцій. Операції за пріоритетом з описом їх позначень, назв та синтаксису наведені у додатку Д.

Відзначимо, що при виконанні операції ділення "/" цілого на ціле у результаті набуваємо цілу частку. Наприклад, `5/2` дорівнює `2`.

Префіксні операції розташовані зліва від операнда і виконують дію з ним до того, як його значення буде використане, а постфіксні операції виконуються після усіх обчислень.

Логічні операції `!`, `&&`, та `||` здійснюють дії над операндами відповідно з таблицею істинності (див. табл. 1.4). Наприклад, при `k=1` результатом виразу `k>0 && k<4` є `true`.

Таблиця 1.4 – Таблиця істинності логічних операцій

Операція	Дія	Вираз	A	B	Результат
!	Логічне НІ ¬	!A	true false		false true
&&	Логічне І ∧	A && B	true true false false	true false true false	true false false false
||	Логічне АБО ∨	A || B	true true false false	true false true false	true true true false

Програма мовою C++ складається з наступних блоків: блоку заголовків програми; блоку з оголошенням класів, прототипів і функцій; головного методу програми; блоку з описом функцій.

Кожна функція, в тому числі і `main`, повинна мати список параметрів. Список може бути порожнім: `()`. За заголовком функції розміщується тіло функції, яке укладене у фігурні дужки. Кожне визначення, опис або оператор закінчується крапкою з комою.

У блоці заголовків програми підключаються зовнішні файли, задається область імен, наводяться директиви препроцесору і вказівки компілятору.

Для підключення до програми зовнішніх файлів (стандартних бібліотек або користувацьких файлів) необхідно використовувати директиву `#include` із зазначенням імені файлу, що підключається.

Директива `#include` включає вміст файлу, шлях до якого заданий, у компільований файл замість рядка з директивою. Якщо шлях поміщений у кутові дужки, то пошук файлу здійснюється в стандартних директоріях. Якщо шлях поміщений у лапки і заданий повністю, то пошук файлу здійснюється в заданій директорії, а якщо шлях повністю не заданий – у поточній директорії. Деякі стандартні бібліотеки наведені нижче:

`iostream` – бібліотека введення-виведення;

`cmath` – математична бібліотека;

`cstring` – бібліотека роботи з рядками.

Для завдання області імен використовується директива `using namespace` із зазначенням конкретної області імен. Для використання стандартної області імен необхідно використовувати конструкцію `using namespace std;`

Заголовки можуть змінюватися в залежності від компілятора, що використовується.

Головний метод програми має стандартну назву `main()`. Виконання програми починається з першого оператора цієї функції. Хоча `main()` і не є ключовим словом, ставитися до нього слід як до ключового, наприклад, не слід використовувати його як ім'я змінної.

Блок з описом функцій містить опис тих функцій, прототипи яких вказані в другому блоці.

Програма може складатися з декількох модулів (вхідних файлів).

Організація виведення даних у консоль виконується за допомогою команди `cout<<`, при цьому необхідно підключити бібліотеку `iostream` та простір імен, до якого належить команда `cout<<`: `using namespace std`. Рядок виводу записують у подвійні лапки. Організація введення даних з консолі виконується за допомогою команди `cin>>`.

В ОС Windows на сьогодні використовується декілька таблиць кодування національних символів, наприклад, cp866, cp1251, utf-8, що інколи призводить до появи незрозумілих символів замість тексту. Для коректного вводу та виводу національних шрифтів на консоль використовують функції `SetConsoleOutputCP()` та `SetConsoleCP()`, в якості параметра обом функціям надається номер кодової сторінки – 1251. Ці функції надані в файлі `<windows.h>`

Приклад виконання роботи

Завдання 1.1. Дані математичні вирази записати мовою С++:

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

Розв'язання

а) `1.0e-3*exp(-2*x)+5*pow(z+1, 3)`

б) `abs(b)<0 && f>1`

Завдання 1.2. Представити математичний запис виразу `0.51e-6*sqrt(3*x)+2*b*b` і показати порядок дій.

Розв'язання

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

3 2 1 6 4 5

Завдання 1.3. Скласти програму обчислення скалярного добутку двох векторів ***a***={4; –2; –4} і ***b***={6; –3; 2}.

1. Постанова задачі

Скласти програму обчислення скалярного добутку двох векторів ***a***={4; –2; –4} і ***b***={6; –3; 2} на мові С++.

2. Методика розв'язання задачі

Скалярний добуток двох векторів обчислюється за формулою:

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

де $A_1, B_1, A_2, B_2, A_3, B_3$ – відповідні координати векторів ***a*** і ***b***.

3. Алгоритм розв'язання задачі

Алгоритм розв'язання задачі можна представити у вигляді такої послідовності дій:

Дія 1. Ввести координати векторів ***a*** і ***b***.

Дія 2. Обчислити скалярний добуток за формулою (1.1).

Дія 3. Вивести значення скалярного добутку двох векторів.

Представимо алгоритм розв'язання задачі на мові С++, позначивши змінні $A_1, B_1, A_2, B_2, A_3, B_3$ і ***ab*** відповідно як `a1`, `b1`, `a2`, `b2`, `a3`, `b3` і `ab` (усі типу `double`).

4. Текст програми

```
//Обчислення скалярного добутку двох векторів
#include <iostream>
#include <cmath>
#include <windows.h>
using namespace std;
int main(){
```

```
  SetConsoleOutputCP(1251);
  SetConsoleCP(1251);
  double a1, a2, a3, b1, b2, b3, ab;
  cout<<"Введіть координати
a1,a2,a3,b1,b2,b3"<<endl;
  cin>>a1>>a2>>a3>>b1>>b2>>b3;
  ab=a1*b1+a2*b2+a3*b3;
  cout<<"Скалярний добуток ab="<<ab<<endl;
return 0;
}
```

5. Результати роботи програми

```
Введіть координати a1,a2,a3,b1,b2,b3
4
-2
-4
6
-3
2
Скалярний добуток ab=22
```

Контрольні питання

1. Які дані можна вживати в мові С++?
2. З якою метою використовують директиву `#include`?
3. Назвіть порядок виконання операцій у виразі.
4. Яка структура програми на мові С++?
5. Як працює оператор присвоєння?
6. Як у С++ здійснюється введення і виведення значень змінних?

Робота № 2
Розробка та реалізація програми з розгалуженою структурою

Мета роботи: оволодіння навичками складання програми з розгалуженою структурою за допомогою умовного оператора `if` або оператора вибору `switch`.

Завдання

Завдання 2.1. Представити математичний запис фрагмента програми за варіантами, які наведені в таблиці 2.1, і обчислити значення змінної `x` після його виконання. Вказівка: замість `n` підставити номер варіанта.

Таблиця 2.1 – Варіанти завдання 2.1

№	Фрагмент програми	№	Фрагмент програми

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;`

Завдання 2.2. Скласти програму обчислення значень функції за варіантами, які наведені в таблиці 2.2, та виконати її.

Таблиця 2.2 – Варіанти завдання 2.2

№	Функція	№	Функція	№	Функція	№	Функція
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	$y = \mathrm{arctg}(1/(\mathrm{x}-1)^2)$	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	$y = \ln \mathrm{tg}x^{1/3}$

Короткі теоретичні відомості

Умовний оператор `if` призначений для виконання або невиконання деякого оператора (простого або складеного) залежно від значення виразу.

Загальні вигляди умовного оператора `if`:

`if(`*вираз*`)` *оператор1;*

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

Виконання умовного оператора `if` полягає в обчисленні логічного *виразу*. Якщо його значення `true` (не дорівнює нулю), то виконується *оператор1*. Якщо значення логічного виразу `false` (дорівнює нулю) і умовний оператор не містить слова `else`, то його виконання завершується. Якщо слово `else` є, то виконується *оператор2*.

Будьте уважні при записі логічного *виразу*: часто замість операції перевірки на рівність == вказується операція присвоєння =, що призводить до некоректної роботи програми. Службові слова `if` та `else` нерозривні, при спробі вставити між ними будь-яку команду виникає помилка на етапі компіляції.

Якщо в якій-небудь гілці умовного оператора треба виконати кілька операторів, то їх слід об'єднати в складений оператор за допомогою операторних дужок `{}`. Один умовний оператор може входити в інший умовний оператор. При цьому кожне слово `else` відповідає останньому перед ним `if`. Так, після виконання наступного фрагмента програми:

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

змінна `y` має значення `10`.

Оператор вибору `switch` призначений для виконання одного з кількох можливих операторів в залежності від збігу значення виразу-селектора та значення константного виразу. Загальний вигляд оператора `switch`:

```
switch (вираз-селектор) {
  case константний вираз 1 : оператор1; break;
       ...
  case константний вираз N : операторN; break;
  default: оператор;
}
```

Вираз-селектор повинен бути цілого типу, типу `char` або типу вказівника. *Константний вираз* повинен мати такий самий тип що

і *вираз-селектор*. В одному операторі `switch` ніякі *константні вирази* не можуть мати однакових значень. Може також бути не більше одного префіксу `default` або він може бути відсутній.

Виконання оператора `switch` починається з обчислення значення *виразу-селектора*. При першому збігові цього значення із значенням *константного виразу* (*1*,..., *N*) виконується відповідний *оператор*. Якщо жодного збігу не зафіксовано, а є слово `default`, то виконується *оператор*, наступний за `default`. У противному разі виконання оператора `switch` завершується.

Слід зазначити, що префікси `case` та `default` самі не змінюють потік управління. Для виходу з оператора `switch` застосовується оператор `break`.

Оператор `break` припиняє виконання оператора `switch`, в якому був виконан цей оператор. Управління передається на оператор, який стоїть одразу за цим оператором `switch`.

У разі, коли необхідно виконати однакові дії для різних значень *константних виразів*, можна записувати декілька виразів поспіль.

Приклад. Після виконання наступного фрагмента програми:

```
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";
}
```

буде надруковано `Case A`

Приклад виконання роботи

Завдання 2.1. Представити математичний запис фрагмента програми

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

і обчислити значення змінної `x` після його виконання.

Розв’язання

Цей фрагмент програми реалізує обчислення функції $x=t^{1/3}$. Після виконання цього фрагмента x=–2.

Завдання 2.2. Скласти програму обчислення значень функції *y=ctgx* і виконати її.

Розв’язання

1. Постановка задачі

Скласти програму обчислення значень функції $y=\operatorname{ctg} x$ на мові C++.

2. Методика розв’язання задачі

Функція $y=\operatorname{ctg} x$ обчислюється за формулою

$$y = \cos x/\sin x, \tag{2.1}$$

якщо

$$\sin x \neq 0. \tag{2.2}$$

3. Алгоритм розв’язання задачі

Алгоритм розв’язання задачі можна представити у вигляді такої послідовності дій:

Дія 1. Ввести значення *x*.

Дія 2. Перевірити умову (2.2). Якщо умова істина, то обчислити значення функції за формулою (2.1) і вивести його, інакше вивести повідомлення: "Функція не існує".

Остаточно представимо алгоритм розв’язання задачі на мові C++, позначивши змінні *x* і *y* відповідно як x і y (обидві типу double).

4. Текст програми

```
//програма обчислення функції y=ctg(x)
#include <iostream>
#include <cmath>
#include <windows.h>
using namespace std;
int main(){
  SetConsoleOutputCP(1251);
  SetConsoleCP(1251);
  double x,y;
  cout<<"Введіть x="; cin>>x;
  if(sin(x)!=0){
     y=cos(x)/sin(x);
     cout<<"y="<<y<<endl;
```

```
    }
  else cout<<"Функція не існує"<<endl;
return 0;
}
```

5. Результати роботи програми

```
Введіть x=1.57
y=0.000796327
```

Контрольні питання

1. Як працюють оператори `if` та `switch`?
2. Коли в операторі `if` застосовують складений оператор?
3. Як виконується умовний оператор `if`, якщо до нього входить інший умовний оператор `if`?
4. Для чого призначений оператор `switch`?
5. Якого типу може бути вираз-селектор в операторі `switch`?
6. Що робить оператор `break` у разі його виконання в операторі `switch`?

Робота № 3
Розробка та реалізація програми з циклічною структурою

Мета роботи: оволодіння навичками складання програми з циклічною структурою за допомогою операторів циклу `while`, `do while` і `for`.

Завдання

Завдання 3.1. Представити математичний запис фрагмента програми за варіантами, які наведені в таблиці 3.1, і обчислити значення змінної `x` після його виконання. Вказівка: замість `n` підставити номер варіанта.

Таблиця 3.1 – Варіанти завдання 3.1

№	Фрагмент програми	№	Фрагмент програми

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;`
			`};`

Завдання 3.2. Скласти програму табулювання функції з завдання 2.2 при зміні значення *x* від –1 до 1 з кроком 0,2 та виконати її.

Короткі теоретичні відомості

Оператор циклу з передумовою `while` складається з ключового слова `while`, за яким йдуть *вираз* логічного типу у круглих дужках та виконуваний у циклі *оператор* (простий чи складений).

Загальний вигляд оператора циклу з передумовою:

`while` (*вираз*) *оператор - тіло циклу*;

Виконання оператора циклу з передумовою починається з обчислення значення *виразу*. Якщо це значення `false`, то *оператор* не виконується, управління передається на оператор, який стоїть одразу за

циклом. Якщо значення *виразу* `true`, *оператор* виконується, після чого знову обчислюється *вираз*. Щоб запобігти зациклюванню, слід передбачити зміну значення *виразу* всередині *тіла циклу*. Наприклад, після виконання наступного фрагмента програми:

```
bool a=true; int x=5;
while(a || x<9){     //цикл завершиться, як тільки
  a=!a; x=x+2;       //вираз набуде значення false
}
```

змінна `x` має значення `11`, а змінна `a` – `0` (`false`).

Оператор циклу з післяумовою `do while` складається з ключового слова `do`, за яким йде виконуваний у циклі *оператор* (простий чи складений); ключового слова `while` і *виразу* логічного типу.

Загальний вигляд оператора циклу з післяумовою:

`do` *оператор - тіло циклу* `while` (*вираз*);

Виконання цього оператора циклу відбувається так. Спочатку виконується *оператор*, а потім визначається значення *виразу* логічного типу. Якщо значення *виразу* `false`, то виконання циклу припиняється. Якщо це значення `true`, то виконується *оператор*, а потім знову обчислюється *вираз*. Наприклад, після виконання наступного фрагмента програми:

```
bool a=true; int x=5;
do {
  a=!a; x=x+2;}      //цикл завершиться, як тільки
while(a || x<9);     //вираз набуде значення false
```

змінна `x` має значення `11`, а змінна `a` – `0` (`false`).

Треба підкреслити, що на відміну від циклу `while`, тіло циклу з післяумовою `do while` завжди виконується хоча б один раз.

Оператор циклу з параметром `for` складається з ключового слова `for`, за яким йдуть у круглих дужках *вираз1*, крапка з комою (`;`), *вираз2*, крапка з комою (`;`), *вираз3* і виконуваний у циклі *оператор* (простий або складений).

Загальний вигляд оператора циклу `for`:

`for` (*вираз1*; *вираз2*; *вираз3*) *оператор - тіло циклу*;

Вираз1 – це вираз, який описує ініціалізацію циклу; *вираз2* – перевірка умови завершення циклу; *вираз3* – це вираз, який обчислюється після

кожної ітерації циклу. Кожний з *виразів1-3* може складатися з декількох виразів, які об'єднуються оператором кома (`,`). Жоден з *виразів1-3* не є обов'язковим.

Виконання оператора циклу `for` починається з обчислення *виразу1*. Потім обчислюється *вираз2*. Якщо значення *виразу2* `false`, то виконання циклу припиняється і управління передається на оператор, який стоїть одразу за циклом. Якщо це значення `true`, то послідовно виконуються *оператор* і *вираз3*, а потім знову обчислюється *вираз2*.

Оператора циклу `for` еквівалентний наступній послідовності операторів:

вираз1;
`while` (*вираз2*) `{`
  оператор; *вираз3*;
`}`

Якщо немає *виразу2*, то вважається, що він має значення `true`. Оператор `for(;;);` представляє собою нескінченний цикл, який еквівалентний оператору `while(true);`.

Приклад. Після виконання наступного фрагмента програми:

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

змінна `x` має значення `11`, а змінна `a` – `0` (`false`).

Оператор `break` припиняє виконання оператора циклу (`while`, `do while` або `for`), в якому був виконаний цей оператор. Управління передається на оператор, який стоїть одразу за оператором циклу.

Оператор `continue` припиняє виконання поточної ітерації оператора циклу (`while`, `do while` або `for`), в якому був виконаний цей оператор, і здійснює перехід до виконання наступної ітерації циклу. Один оператор циклу може входити в інший оператор циклу.

При виконанні розрахунків над дійсними числами виникає похибка. Треба знати способи оцінки такої похибки та вміти застосовувати принципи роботи з наближеними значеннями. Більш детально оцінку похибки при роботі з дійсними числами розглянуто в Додатку И.

Приклад виконання роботи

Завдання 3.1. Представити математичний запис фрагмента програми

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

й обчислити значення змінної `x` після його виконання.

Розв’язання

Цей фрагмент програми реалізує обчислення $x!=1\cdot2\cdot...\cdot5$. Після виконання цього фрагмента x=120.

Завдання 3.2. Скласти програму табулювання функції $y = \operatorname{ctg} x$ при зміні значення x від $a=-1$ до $b=1$ з кроком $h=0{,}5$ і виконати її.

Розв’язання

1. Постановка задачі

Скласти програму табулювання функції $y = \operatorname{ctg} x$ при зміні значення x від $a=-1$ до $b=1$ з кроком $h=0{,}5$ на мові С++.

2. Методика розв’язання задачі

Методика розв’язання задачі збігається з методикою з прикладу завдання 2.2 (при оформленні роботи треба навести методику наново).

3. Алгоритм розв’язання задачі

Алгоритм розв’язання задачі можна представити у вигляді такої послідовності дій:

Дія 1. Ввести значення a, b, h.

Дія 2. Присвоїти x значення a.

Дія 3. Повторювати наступні дії поки $x \le b$:

Дія 3.1. Перевірити умову (2.2). Якщо умова істинна, то обчислити значення функції y за формулою (2.1) і вивести x та y, інакше вивести x та повідомлення: ’не існує’;

Дія 3.2. Присвоїти x нове значення, яке дорівнює старому значенню x плюс крок h.

Запишемо алгоритм розв’язання задачі мовою С++, позначивши змінні x, y, a, b, h відповідно як x, y, a, b, h (усі типу double).

4. Текст програми

```
#include <iostream.h>
#include <cmath.h>
#include <windows.h>
using namespace std;
int main(){
  SetConsoleOutputCP(1251);
  SetConsoleCP(1251);
  double a,b,h,x,y;
  cout<<"Введіть 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 існує"<<endl;
return 0;
}
```

5. Результати роботи програми

```
Введіть 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
```

Контрольні питання

1. Як працюють оператори циклу `while`, `do while` і `for`?
2. Чим цикл `do while` відрізняється від циклу `while`?
3. Коли у циклах `while` та `for` застосовують складений оператор?
4. Які обов'язкові елементи присутні у циклі `for`?
5. Якій послідовності операторів еквівалентний оператор циклу `for`?
6. Як працюють оператори `break` та `continue`?

Робота № 4
Розробка та реалізація програми з використанням функцій

Мета роботи: оволодіння навичками складання програми з використанням функцій та виконання її.

Завдання

Завдання 4.1. Представити математичний запис фрагмента програми за варіантами, які наведені в таблиці 4.1, та обчислити значення змінних, які будуть виведені на екран. Вказівка: n дорівнює номеру варіанта.

Таблиця 4.1 – Варіанти завдання 4.1

№	Фрагмент програми	№	Фрагмент програми

1-5	int f(int a){
	int t=1;
	for (int i=1; i<=a; i++)
	t*=i;
	return t;
	}
	int fi(int k){
	int t=1;
	for (int i=1;i<=k;i++)
	t+=f(i);
	return t;
	}
	int main()
	{ cout << fi(n)<<"\n";
	return 0;
	}
		6-10	double f1(int a){
			double t=a;
			for (int i=1; i<a; i++)
			t=a+sqrt(t);
			return t;
			}
			double f2(int a){
			double t=a;
			for (int i=1;i<a;i++)
			t=a-
			pow(t,1.0/(2*i+1));
			return t;
			}
			int main()
			{
			cout << f1(n)/f2(n%3+4);
			return 0;
			}

11-15	double f(int a){
	double t=1,f=1,r=1;
	for (int i=1; i<=a; i++)
	{
	f*=a*a;
	t*=(2*i)*(2*i+1);
	r+=f/t;
	}
	return r;
	}
	int main()
	{ cout <<
	f(n%4+4)<<"\n";
	return 0;
	}
		16-20	bool f(int a){
			bool s=true;
			for (int i=3;
			i<=sqrt(a); i+=2)
			if (a%i==0)
			return false;
			return s;
			}
			int main()
			{ cout << "2 ";int k=1;
			for (int i=3;k<n;i+=2)
			if (f(i)){cout<<i<<"
			";
			k++;}
			return 0;
			}

21-25	int f(int a){
	int f,s=0;
	for(int t=a;t>0;t/=10){
	f=t%10;
	s+=f;
	}
	return s;
	}
	int main()
	{ int t;t=n%3+3;
	cout << f(pow(n,t))
	<<"\n";
	return 0;
	}
		26-30	int f(int a){
			int s=0;
			for(int t=a;t>0;t/=10){
			s++;
			}
			return s;
			}
			int main()
			{
			int k;k=n%3+3;
			cout << f(pow(n,k))
			<<"\n";
			return 0;
			}

Завдання 4.2. Скласти програму обчислення величин із завдання 3.2 з використанням функцій і виконати її.

Короткі теоретичні відомості

Функція – спеціальна конструкція, за допомогою якої фрагмент коду, що повторюється в програмі кілька разів, виноситься за тіло основної програми.

Існує два способи оголошення функції: до функції `main`; за допомогою прототипу (тіло оголошеної функції описується після функції `main`). При другому способі необхідно до функції `main` вказати тип значення, що повертається, ім'я функції та її аргументи.

Загальний синтаксис оголошення функції:

тип_значення ім'я_функції (параметри) {

тіло функції;

}

Значення, що повертається – результат роботи функції, може бути будь-яким з базових або користувацьких типів. Якщо функція нічого не повертає, на місце значення, що повертається, підставляється `void` (пусто).

Параметри (аргументи) – вхідні дані, необхідні для роботи функції, для кожного параметра вказують тип даних. Параметри можуть бути відсутніми.

Параметри, які вказуються при визначенні функції, називаються формальними, вони створюються в момент виклику функції в оперативній пам'яті. При виході з функції такі параметри будуть

знищені. Параметри, які передаються в функцію при її виклику, називаються фактичними.

Формальному параметру функції може бути задане значення за замовчуванням, параметрами за замовчуванням можуть бути параметри, починаючи з правого кінця списку без перерв:

тип_значення ім'я_функції (тип_параметра ім'я _параметра = значення_за_замовчуванням);

Не можна створювати одну функцію всередині іншої і викликати функцію до її оголошення.

Для повернення значення з функції в програму використовується оператор `return`:

```
return res;
```

Якщо функція не повертає жодних значень, оператор `return` можна використовувати для зупинки функції, в даному випадку `return` відпрацює для функції, як `break` для циклу. Операторів `return` в функції може бути декілька, але спрацює тільки один з них. Якщо тип, що повертається функцією, не `void`, то необхідно завжди використовувати форму: `return` *значення;*

Виклик функції складається з зазначення імені функції, передачі аргументів (при необхідності) і отримання значення, що повертається (при необхідності). При виклику функції необхідно вказувати таку кількість параметрів, яка була визначена при оголошенні функції.

При використанні масиву в якості аргументу функції ім'я масиву перетворюється до вказівника на його перший елемент, тобто при передачі масиву в якості аргументу функції відбувається передача вказівника. При передачі одновимірного масиву досить вказати порожні квадратні дужки:

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

При передачі двовимірного масиву обов'язково необхідно вказати кількість стовпців, кількість рядків вказувати не обов'язково:

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

Будь-які операторні дужки в програмному коді утворюють так звану область видимості. Змінні, оголошені всередині цих дужок, буде видно тільки в цій області видимості. Згідно з правилами області видимості, змінні діляться на два види – локальні і глобальні.

Локальні змінні створюються всередині будь-якої області видимості. Глобальні змінні створюються поза всяких областей видимості, переважно до функції `main()`. Глобальна змінна видна в будь-якому місці програми. За замовчуванням глобальні змінні на відміну від локальних ініціалізуються 0. Ті зміни, які відбуваються з глобальною змінною всередині функції, при виході з останньої зберігаються.

Якщо є глобальна і локальна змінні з однаковими іменами, то всередині будь-якої області видимості буде використовуватися локальна змінна. Тому слід уникати використання в програмі однакових імен змінних.

Якщо функція викликає сама себе, то вона називається рекурсивною. Глибина рекурсії, тобто кількість викликів, у C++ не обмежується. Реально вона залежить від ресурсів пам'яті (розміру стека).

В загальному випадку рекурсивність – це не властивість самої функції, а властивість її опису. Рекурсивна функція, як правило, коротша і наочніша за нерекурсивну, але при виконанні вимагає більше часу і пам'яті за рахунок повторних звертань до самої себе і дублювання локальних змінних. Необхідно добре розуміти, що кожний черговий рекурсивний виклик приводить до утворення нової копії локальних об'єктів функції і усі ці копії, що відповідають ланцюжкові активізованих і незавершених рекурсивних викликів, існують незалежно один від одного та зберігаються у стеку. Це може привести до так званого «вибуху» стека. «Вибух» стека означає, що для запису локальних змінних функції не вистачає пам'яті, яка відводиться під стек.

Приклад виконання роботи

Завдання 4.1. Представити математичний запис та обчислити значення змінних, які будуть виведені на екран, після виконання наступного фрагмента програми:

```
long fact(long x);
int main(){
  long n,x,y; n=40; n-=36;
  y=fact(n); x=n; cout<<x<<" "<<y;
  return 0;
}
long fact(long x){
  long f=1;
```

```
   for(long i=2;i<=x;f*=i++);
   return f;
 }
```

Розв’язання

Ця програма обчислює *x*!=4∗3∗2∗1. У результаті її виконання x=4, а y=24.

Завдання 4.2. Скласти програму табулювання функції *y=ctgx* при зміні значення *x* від *a*=−1 до *b*=1 з кроком *h*=0,5. При розробці програми використовувати функції та виконати її.

Розв’язання

1. Постановка задачі

Скласти програму табулювання функції *y* = ctg*x* при зміні значення *x* від *a*=−1 до *b*=1 з кроком *h*=0,5 на мові С++ з використанням функцій.

2. Алгоритм розв’язання задачі

Алгоритм наведений у прикладі розв’язання завдання 3.2, при оформленні роботи алгоритм необхідно навести наново.

3. Текст програми

```
// програма табулювання функції
#include <iostream>
#include <cmath>
#include <windows.h>
using namespace std;
void tab(double a,double b, double h);

int main(){
  SetConsoleOutputCP(1251);
  SetConsoleCP(1251);
  double a,b,h;
  cout<<"Введіть a,b,h: "; cin>>a>>b>>h;
  tab(a,b,h);
  return 0;
}
//функція табулювання
void tab(double a,double b, double h){
   double x,y;
   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 існує"<<endl;
}
```

4. Результати роботи програми

```
Введіть 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
```

Контрольні питання

1. Які існують способи оголошення функції?
2. Яка різниця між фактичними та формальними параметрами?
3. Як працює оператор `return`?
4. Що таке глобальна та локальна змінні?
5. Як здійснюється виклик функції?

Робота № 5
Розробка та реалізація програми з масивами

Мета роботи: оволодіння навичками складання програми з масивами та виконання її.

Завдання

Завдання 5.1. Представити математичний запис фрагмента програми та обчислити значення змінної x після його виконання. Елементи масиву обчислюються за формулою $a_i = b_i - 32$, де $b_{i+1}=(37 \cdot b_i+3) \bmod 64$. Значення b_0 дорівнює номеру варіанта.

Таблиця 5.1 – Варіанти завдання 5.1

№	Фрагмент програми	№	Фрагмент програми

1-5	`t=2; n=3; x = a[0];`
	`for(int j=0;j<n;j++){`
	`  x=x*t+a[j+1];`
	`}`
		6-10	`n=4; x = a[n];`
			`for(int j=n-1;j>=0;j--`
			`){`
			`  x=a[j]+1/x;`
			`}`

11-15	`n=4; x = a[0];`
	`for (int j=1;j<n;j++){`
	`  if (a[j]>x) x=a[j];`
	`}`
		16-20	`t=3; n=3; x=a[n];`
			`for (int j=0;j<n;j++){`
			`  x=x+a[j]*pow(t,n-j);`
			`}`

21-25	`n=4; m=n/2; k=n-1;`
	`for (int j=0;j<m;j++){`
	`  y=a[j];a[j]=a[k];`
	`  a[k]=y; k--;`
	`}`
	`x=a[0];`
		26-30	`n=4; k=0; x=0;`
			`for (int j=0;j<n;j++)`
			`  if(a[j]>0){`
			`  x+=a[j];k++;`
			`  }`
			`if (k!=0) x/=k;`

Завдання 5.2. Скласти програму обчислення наступних величин за варіантами, які наведені в таблиці 5.2, та виконати її, якщо елементи масиву визначаються за формулою $a_i = b_i - 32$, де $b_{i+1}=(37 \cdot b_i+3) \bmod 64$. Значення b_0 дорівнює номеру варіанта; i змінюється від 0 до 18.

Таблиця 5.2 – Варіанти завдання 5.2

№	Величини, які потрібно обчислити
1-3	Найбільший елемент масиву ***a*** і його порядковий номер
4-6	Суму елементів масиву ***a***, значення яких кратні номеру варіанта
7-9	Суму елементів масиву ***a***, значення яких парні числа
10-12	Середнє арифметичне додатних елементів масиву ***a***
13-15	Суму елементів масиву ***a***, значення яких непарні числа
16-18	Середнє геометричне додатних елементів масиву ***a***

19-21	Суму елементів масиву ***a***, значення яких двозначні парні числа
22-24	Добуток найбільшого і найменшого елементів масиву ***a***
25-27	Суму елементів масиву ***a***, значення яких двозначні непарні числа
28-30	Модуль вектора ***a***/3

Короткі теоретичні відомості

Масив – це упорядкована послідовність однойменних елементів, кожен з яких має один і той самий тип. Елементи масиву можуть мати любий стандартний тип або тип, введений користувачем. Елементи масиву розміщені впорядковано, кожен має свій номер, який називається індексом. Доступ до елементів масиву відбувається шляхом вказування імені масиву та порядкового номера елемента (індексу).

Масив визначається за модифікатором типу `[]`. Загальний вигляд опису масиву:

тип_елементів ім'я_масиву`[`*кількість_елементів*`];`

Наприклад:

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

визначає масив з двадцяти елементів типу `char`. Ім'я масиву `sName` містить адресу першого елементу масиву. Це ім'я може бути використане в операціях арифметики вказівників для доступу до елементів масиву.

Оператор `[]` дає більш короткий вираз для доступу до елементів масиву. Вираз `sName[k]` є еквівалентним `*(sName+k)`. Більш детально робота з оператором `*` (розіменування) буде описана у частині 2 даних методичних вказівок.

Масиви в C++, як і в C, нумеруються з нуля. Тому найбільший елемент – це *кількість_елементів* мінус одиниця. В масиві з 20 елементів індекси змінюються від 0 до 19. Перехід через максимум призводить до використання чужої області пам'яті та помилки на етапі виконання програми.

Елементи масиву розміщуються у пам'яті послідовно. Елементи з меншими значеннями індексу зберігаються в більш низьких адресах пам'яті.

Операції з масивами зручніше проводити за допомогою циклів. Неможливо присвоїти один масив іншому цілком, тільки поелементно.

Приклад виконання роботи

Завдання 5.1. Представити математичний запис фрагмента програми та обчислити значення змінної *x* після його виконання. Елементи масиву обчислюються за формулою $a_i = b_i - 32$, де $b_{i+1}=(37 \cdot b_i+3) \bmod 64$. Значення b_0 дорівнює 35.

```
int n=4; int x=a[0];
for (int i=1;i<n;i++)
  if (fabs(a[i])<x)  x=fabs(a[i]);
```

Розв'язання

Обчислимо елементи масиву:

$b_0 = 35$

$a_1 = 35-32=3;$

$b_1 = (37 \times b0+3) \bmod 64=(37 \times 35+3) \bmod 64=18$

$a1 = 18-32=-14;$

$b2 = (37 \times b1+3) \bmod 64=(37 \times 18+3) \bmod 64=29$

$a2 = 29-32=-3;$

$b3 = (37 \times b2+3) \bmod 64=(37 \times 29+3) \bmod 64=52$

$a_3 = 52-32=20.$

Цей фрагмент програми знаходить мінімальний за абсолютним значенням елемент масиву (вектора):

$a=\{3;-14;-3;20\}$, $x=min|a_i|$, $(i=0,1,2,3)$

Після виконання фрагменту $x=-14$.

Завдання 5.2. Скласти програму перестановки елементів масиву ***a*** в зворотному порядку і виконати її, якщо елементи масиву визначаються за формулою $a_i = b_i - 32$, де $b_{i+1}=(37 \cdot b_i+3) \bmod 64$. Значення $b_0=35$; *i* змінюється від 0 до 16.

Розв'язання

1. Постановка задачі

Скласти програму перестановки елементів масиву ***a*** в зворотному порядку на мові С++, якщо елементи масиву визначаються за формулою $a_i = b_i - 32$, де $b_{i+1}=(37 \cdot b_i+3) \bmod 64$. Значення $b_0=35$; *i* змінюється від 0 до 16.

2. Алгоритм розв'язання задачі

Алгоритм розв'язання задачі можна представити у вигляді такої послідовності дій:

Дія 1. Обчислити елементи масиву ***b***.

Дія 2. Обчислити елементи масиву ***a***.

Дія 3. Вивести елементи масиву ***a***.

Дія 4. Визначити *m* (кількість перестановок елементів масиву ***a***), як результат цілочислового ділення числа елементів масиву *n* на 2.

Дія 5. Присвоїти *j* (номеру поточного елемента масиву, який переставляється на місце елемента з меншим номером) значення *n*-1.

Дія 6. Повторювати *m* разів наступні дії (*i*= 0, 1, ..., *m*-1):

Дія 6.1. Присвоїти *x* (змінній для тимчасового зберігання елемента з меншим номером) значення *i*-го елемента масиву ***a***;

Дія 6.2. Присвоїти *i*-му елементу масиву ***a*** значення *j*-го елемента;

Дія 6.3. Присвоїти *j*-му елементу масиву ***a*** значення *x*;

Дія 6.4. Зменшити значення *j* на 1;

Дія 7. Вивести елементи масиву ***a*** після перестановки.

Запишемо алгоритм розв'язання задачі мовою С++, позначив масив ***a*** через `a`, елементи якого мають тип `int`. Змінні *i*, *j*, *n*, *m*, *x* позначимо відповідно як `i`, `j`, `n`, `m`, `x` (усі мають тип `int`).

3. Текст програми

```
//програма перестановки елементів масиву a[n]
#include <iostream>
#include <windows.h>
using namespace std;
void rev(int a[],int n);
void printArray(int a[],int n,char str[]);
int main(){
  SetConsoleOutputCP(1251);
  SetConsoleCP(1251);
  int n=17;
  int a[17],b[17];
cout<<"Обчислюємо масив a["<<n<<"]"<<endl;
b[0]=35;
for(int i=0; i<n-1; i++)b[i+1]=(37*b[i]+3)%64;
for(int i=0; i<n; i++)a[i]=b[i]-32;

printArray(a,n,"Масив a до перестановки");
rev(a,n);
printArray(a,n,"Масив a після перестановки");
return 0;
```

```
}
//функція перестановки елементів масиву int a[n]
void rev(int a[],int n){
  int i,j,x;
  for(i=0,j=n-1; i<n/2; i++,j--){
     x=a[i]; a[i]=a[j]; a[j]=x;
  }
}
//функція друкування елементів масиву
void printArray(int a[], int n,char str[]){
    cout<<str<<endl;
    for(int i=0; i<n; i++)
        cout<<a[i]<<" ";
    cout<<endl;
}
```

4. Результати роботи програми

```
Обчислюємо масив a[17]
Масив a до перестановки
3 -14 -3 20 -25 -26 1 -24 11 26 5 -4 -17 14 9 16 19
Масив a після перестановки
19 16 9 14 -17 -4 5 26 11 -24 1 -26 -25 20 -3 -14 3
```

Контрольні питання

1. Який тип можуть мати елементи масиву?
2. Як відбувається доступ до елементів масиву?
3. Який індекс мають найменший та найбільший елементи масиву?
4. Як розміщуються в пам’яті елементи масиву?
5. Як зростають індекси багатовимірних масивів?
6. Як можна присвоїти значення елементів одного масиву іншому?

Робота № 6
Розробка та реалізація програм пошуку та сортування

Мета роботи: оволодіння навичками складання програми пошуку та сортування елементів масиву та виконання її.

Завдання

Завдання 6.1. . Представити математичний запис фрагмента програми та обчислити значення змінної *x* після його виконання, згідно варіанту, наведеному в таблиці 6.1. Елементи масиву обчислюються за формулою $a_i = p_i - 50$, де $p_{i+1}=(11 \cdot p_i+7) \bmod 100$. Значення p_0 дорівнює номеру варіанта, кількість елементів у масиві дорівнює `size=15`.

Таблиця 6.1 – Варіанти завдання 6.1

№	Фрагмент програми

1–5	`for (int i=0; i< size; i++) {`
	`  for (int j= size-1; j>i; j--){`
	`  if (a[j] < a[j-1]) {`
	`  int t = a[j];`
	`  a[j] = a[j-1];`
	`  a[j-1] = t;`
	`  }`
	`  }`
	`  }`
	`x=a[0];`

6–10	`for (int i=1; i<size; i++) {`
	`  int curr = a[i];`
	`  int j = i-1;`
	`  while (j>=0 && a[j]>curr) {`
	`  a[j+1]=a[j];`
	`  j--;`
	`  }`
	`  a[j+1] = curr;`
	`  }`
	`x=a[0];`

11–15	`  for (int i=0; i<size-1; i++) {`
	`  int nmin = i;`
	`  for (int j=i+1; j<size; j++){`
	`  if (a[j]<a[nmin]) nmin = j;`
	`  }`
	`  if (i != nmin) {`
	`  int t = a[nmin];`
	`  a[nmin] = a[i];`
	`  a[i] = t;`
	`  }`
	`  }`
	`x=a[0];`

16–20	`for (int i=0; i< size; i+=2){`
	`  for (int j= size-1; j>i; j-=2){`
	`  if (a[j] > a[j-2] ) {`
	`  int t = a[j];`
	`  a[j] = a[j-2];`
	`  a[j-2] = t;`
	`  }`
	`  }`
	`  }`
	`x=a[2];`

21–25	`for (int i=0; i<size-1; i++) {`
	`  int nmin = i;`
	`  for (int j=i+1; j<size; j++){`
	`  if (a[j]>a[nmin]) nmin = j;`
	`  }`
	`  if (i != nmin) {`
	`  int t = a[nmin];`
	`  a[nmin] = a[i];`
	`  a[i] = t;`
	`  }`
	`}`
	`x=a[0];`

26–30	`for (int i=1; i<size; i+=2) {`
	`  int curr = a[i];`
	`  int j = i-2;`
	`  while (j>=0 && a[j]>curr) {`
	`  a[j+2]=a[j];`
	`  j-=2;`
	`  }`
	`  a[j+2] = curr;`
	`  }`
	`x=a[1];`

Завдання 6.2. Скласти програму сортування послідовностей чисел згідно з варіантами, які наведені в таблиці 6.2, та виконати її. Послідовності заповнити за допомогою датчика випадкових чисел.

Таблиця 6.2 – Варіанти завдання 6.2

№	Завдання
1	Дана послідовність цілих чисел $a_1, a_2, \dots, a_{25}$. Розташувати елементи послідовності, кратні 3, за зростанням (інші елементи залишаються на своїх місцях). Метод сортування – вибір.
2	Дана послідовність дійсних чисел $a_1, a_2, \dots, a_{22}$. Розташувати додатні елементи послідовності, що стоять на непарних місцях за зростанням (інші елементи залишаються на своїх місцях). Метод сортування – вибір.
3	Дана послідовність дійсних чисел $a_1, a_2, \dots, a_{22}$. Елементи, що стоять

	на парних місцях, розташувати в порядку зростання, а на непарних – в порядку зменшення. Метод сортування – бульбашкою.
4	Дана послідовність дійсних чисел $a_1, a_2, \dots, a_{22}$. Розташувати елементи послідовності, що стоять на парних місцях, у порядку зменшення (інші елементи залишаються на своїх місцях). Метод сортування – бульбашкою.
5	Дана послідовність дійсних чисел $a_1, a_2, \dots, a_{22}$. Розташувати елементи послідовності, менші за середньоарифметичне значення, за зростанням (інші елементи залишаються на своїх місцях). Метод сортування – вставка.
6	Дана послідовність дійсних чисел $a_1, a_2, \dots, a_{22}$, відмінних від нуля. Розташувати від'ємні елементи послідовності за спаданням, а додатні – за зростанням. Метод сортування – вибір.
7	Дана послідовність додатних цілих чисел $a_1, a_2, \dots, a_{25}$. Розташувати елементи послідовності, що кратні 3, за спаданням (інші елементи залишаються на своїх місцях). Метод сортування – вставка.
8	Дана послідовність цілих чисел $a_1, a_2, \dots, a_{25}$. Розташувати парні елементи за зростанням (інші елементи залишаються на своїх місцях). Метод сортування – вибір.
9	Дана послідовність дійсних чисел $a_1, a_2, \dots, a_{22}$. Потрібно розташувати додатні елементи послідовності за спаданням (інші елементи залишаються на своїх місцях). Метод сортування – вставка.
10	Дана послідовність цілих чисел $a_1, a_2, \dots, a_{25}$. Розташувати непарні елементи за спаданням (інші елементи залишаються на своїх місцях). Метод сортування – вибір.
11	Дана послідовність дійсних чисел $a_1, a_2, \dots, a_{22}$. Розташувати елементи послідовності, які належать діапазону $[p_1, p_2]$, за спаданням (інші елементи залишаються на своїх місцях). Метод сортування – бульбашкою.
12	Дана послідовність дійсних чисел $a_1, a_2, \dots, a_{30}$. Розташувати першу половину елементів послідовності за спаданням, інші елементи розташувати за зростанням. Метод сортування – бульбашкою.
13	Дана послідовність дійсних чисел $a_1, a_2, \dots, a_{22}$. Розташувати елементи послідовності, які містять цифру N, за зростанням (інші елементи залишаються на своїх місцях). Метод сортування – вставка.
14	Дана послідовність дійсних чисел $a_1, a_2, \dots, a_{20}$. Розташувати елементи послідовності, сума цифр яких дорівнює N, за спаданням (інші елементи залишаються на своїх місцях) Метод сортування – вибір.
15	Дана послідовність цілих чисел $a_1, a_2, \dots, a_{30}$. Розташувати додатні елементи послідовності, які кратні 5, за спаданням (інші елементи залишаються на своїх місцях). Метод сортування – вибір.

Короткі теоретичні відомості

Найбільш простий зі способів пошуку даних – лінійний пошук. Даний алгоритм порівнює кожний елемент масиву із ключем, наданим для пошуку. Алгоритм лінійного пошуку відмінно працює тільки для невеликих або неупорядкованих масивів і є абсолютно надійним.

Якщо масив містить упорядковану послідовність даних, то більш ефективним буде двійковий (бінарний) пошук.

Припустимо, що змінні *Lb* і *Rb* містять, відповідно, ліву й праву границі відрізка масиву, де перебуває потрібний елемент. Пошук завжди будемо починати з аналізу середнього елемента *M* відрізка масиву. Якщо шукане значення менше *M*, переходимо до пошуку в лівій половині відрізка, де всі елементи менші за елемент, який тільки що перевірили. Інакше кажучи, значенням *Rb* стає (*M*-1) і на наступній ітерації робота ведеться з половиною масиву. Таким чином, у результаті кожної перевірки вдвічі звужується область пошуку.

Двійковий пошук – дуже потужний метод, наприклад, якщо довжина масиву дорівнює 1023, тоді після першого порівняння проміжок звужується до 511 елементів, а після другого – до 255, тобто для пошуку в масиві з 1023 елементів досить 10 порівнянь.

Ідея методу бульбашкового сортування полягає в наступному: крок сортування полягає в проході знизу вгору по масиву. По дорозі проглядаються пари сусідніх елементів. Якщо елементи деякої пари знаходяться в неправильному порядку, то вони міняються місцями. Після першого проходу по масиву "вгорі" виявляється самий "легкий" елемент. Наступний прохід робиться до другого зверху елемента, таким чином другий за величиною елемент піднімається на правильну позицію. Перегляд елементів виконується по "нижній" частині масиву, яка кожного разу зменшується до тих пір, доки в ній не залишиться тільки один елемент. На цьому сортування закінчується, оскільки послідовність впорядкована за зростанням.

Основні принципи методу: середня кількість порівнянь та перестановок мають квадратичний порядок зростання, звідси можна зробити висновок, що алгоритм бульбашки дуже повільний і малоефективний. Проте, у нього є величезний плюс: він простий і його можна по-всякому покращувати.

Якщо на якомусь із проходів не відбулося жодного обміну, це означає, що всі пари розташовані в правильному порядку, тобто масив

вже відсортований. Таким чином перший крок оптимізації полягає в запам'ятовуванні, чи проводився при даному проході будь-який обмін. Якщо ні – алгоритм закінчує роботу.

Ідея методу сортування вибором полягає в тому, щоб створювати відсортовану послідовність шляхом приєднання до неї одного за одним елементів у правильному порядку. Алгоритм складається з n послідовних кроків, починаючи від нульового і закінчуючи (n-1). На i-му кроці вибираємо найменший з елементів $a[i]$ ... $a[n]$ і міняємо його місцями з $a[i]$. Незалежно від номера поточного кроку i, послідовність $a[0]$ ... $a[i]$ є впорядкованою. Таким чином, на етапі (n-1) вся послідовність, крім $a[n]$ елемента виявляється відсортована, а $a[n]$ стоїть на останньому місці по праву: все менші елементи вже пішли вліво.

Основні принципи методу: для знаходження найменшого елемента з n+1 розглядаємих елементів алгоритм робить n порівнянь. Оскільки кількість перестановок завжди буде меншою за кількість порівнянь, час сортування зростає щодо кількості елементів. Крім того, алгоритм не використовує додаткову пам'ять: всі операції відбуваються "на місці".

Сортування цим методом можна використовувати для масивів, що мають невеликі розміри.

Сортування простими вставками в чомусь схоже на методи, наведені раніше. Аналогічним чином робляться проходи по частині масиву, й аналогічним же чином на його початку "виростає" відсортована послідовність.

Однак у сортуванні бульбашкою або вибором можна було чітко заявити, що на i-му кроці елементи $a[0]$ ... $a[i]$ стоять на правильних місцях і нікуди вже не переміститься. В цьому ж сортуванні подібне твердження буде більш слабким: послідовність $a[0]$ ... $a[i]$ впорядкована. При цьому по ходу алгоритму в неї будуть вставлятися нові елементи.

Розглянемо дії алгоритму на i-му кроці. Послідовність до цього моменту розділена на дві частини: впорядковану $a[0]$ ... $a[i]$ та невпорядковану $a[i+1]$ ... $a[n]$. На наступному, (i+1)-му кожному кроці алгоритму беремо *a[i+1]*-й елемент та вставляємо на потрібне місце у відсортовану частину масиву. Пошук відповідного місця для чергового елементу вхідної послідовності здійснюється шляхом послідовних порівнянь з елементом, що стоять перед ним. Залежно від результату порівняння елемент або залишається на поточному місці (вставка завершена), або вони міняються місцями і процес повторюється.

Приклад виконання роботи

Завдання 6.1. Представити математичний запис фрагмента програми та обчислити значення змінної *x* після його виконання, де `size=22` – розмір масиву, `N=31` – варіант за списком групи.

```
for (int i=0; i<size-1; i++) {
      if(a[i]%2==0)
      {int nmin = i;
        for (int j=i+1; j<size; j++){
            if (a[j]<a[nmin]&& (a[j]%2==0)) nmin = j;
        }
        if (i != nmin) {
            int t = a[nmin];
            a[nmin] = a[i];
            a[i] = t;
        }
    }
      }
x=a[3];
```

Розв’язання

Даний фрагмент програми виконує сортування парних елементів масиву за зростанням. Реалізація сортування виконана методом вибору, зміна `x=-38`.

Завдання 6.2. Дана послідовність цілих чисел $a_1, a_2, \dots, a_{35}$. Розташувати елементи послідовності, кратні 5 за зростанням (інші елементи залишаються на своїх місцях). Сортування виконати методом вибору.

Розв’язання

1. Постановка задачі

Дана послідовність цілих чисел $a_1, a_2, \dots, a_{35}$. Відсортувати за зростанням тільки елементи, які кратні 5.

2. Алгоритм розв’язання задачі

Алгоритм розв’язання задачі можна представити у вигляді такої послідовності дій:

Дія 1. Ввести елементи одновимірного масиву *a*.

Дія 2. Відсортувати елементи, які кратні 5.

Дія 3. Вивести впорядкований масив на екран.

3. Текст програми

```
#include <iostream>
#include <windows.h>
```

```
using namespace std;
void chooseSort(int a[], int size){
for (int i=0; i<size-1; i++) {
    if(a[i]%5==0)
     {int nmin = i;
     for (int j=i+1; j<size; j++){
        if (a[j]<a[nmin]&& (a[j]%5==0)) nmin = j;
        }
        if (i != nmin) {
            int t = a[nmin];
            a[nmin] = a[i];
            a[i] = t;
        }
    }
      }
}
void printArr(int a[], int size){
    for(int i=0; i<size; i++) {
        cout << a[i] << " ";
    }
    cout << "\n";
}
int main(){
   SetConsoleCP(1251);SetConsoleOutputCP(1251);
   int p[22], arr[22],n=22;
   p[0]=31;
   for(int i=0;i<n-1;i++)
   p[i+1]=(p[i]*11 + 7) % 100;
      for(int i=0;i<n;i++)
        arr[i]=p[i]-50;
  cout<<"До сортування:\n";
    printArr(arr, n);
    chooseSort(arr, n);
  cout<<"Після сортування:\n";
    printArr(arr, n);
    return 0;
}
```

4. Результати роботи програми

```
До сортування
-19 -2 -15 42 -31 -34 33 -30 -23 -46 1 18 5 -38 -11
-14 -47 -10 -3 -26 21 38
Після сортування
-19 -2 -30 42 -31 -34 33 -15 -23 -46 1 18 -10 -38 -
11 -14 -47 5 -3 -26 21 38
```

Контрольні питання

1. Який основний принцип роботи алгоритму лінійного пошуку?
2. До якого масиву можна застосовувати алгоритм бінарного пошуку?
3. Який основний принцип роботи алгоритму бінарного пошуку?
4. Який основний принцип роботи алгоритму бульбашкового сортування?
5. Як працює алгоритм сортування вибором?
6. Як працює алгоритм сортування простими вставками?

Робота № 7
Розробка та реалізація програми з використанням рядків

Мета роботи: оволодіння навичками складання програми з використанням рядків та виконання її.

Завдання

Завдання 7.1. Визначити дію фрагмента програми за варіантами, які наведені в таблиці 7.1, якщо `char str1[20]="C++ language"; const char *str2="12345"; int n;` (Вказівка: `n` дорівнює номеру варіанта).

Таблиця 7.1 – Варіанти завдання 7.1

№	Фрагмент програми	№	Фрагмент програми
1-5	`strncat(str1,str2,n); 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;`

Завдання 7.2. Скласти програму знаходження наступних величин за варіантами, які наведені в таблиці 7.2, та виконати її, якщо дано рядок "We study C++ programming language first semester.".

Таблиця 7.2 – Варіанти завдання 7.2

№	Величини, які потрібно знайти
1-3	Кількість слів у рядку
4-6	Кількість букв e у рядку
7-9	Довжину найкоротшого слова у рядку
10-12	Довжину найдовшого слова у рядку
13-15	Кількість заголовних букв у рядку
16-18	Перші букви кожного слова
19-21	Слова, які не містять букву e
22-24	Які букви і скільки раз зустрічаються у рядку
25-27	Яких букв більше у рядку: голосних або приголосних
28-30	Позиції символів, які не є буквами

Короткі теоретичні відомості

У мові C ++, як і в мові C, немає вбудованої підтримки строкового типу даних, для роботи з рядками використовуються символьні масиви, в кожному елементі зберігається один символ і займає один байт, тому що кожен елемент символьного масиву має тип `char`. Ознакою кінця

рядка є нуль-термінатор \0 (нульовий символ). Нуль-символ – це не цифра 0, в таблиці кодів ASCII він має номер 0 та не виводиться на друк. Наявність нуль-символу означає, що кількість елементів масиву має бути хоча б на один більше, ніж кількість символів у рядку. При використанні у виразах рядок укладається в подвійні лапки. Лапки не є частиною рядка, вони відзначають його початок і кінець.

Варіанти ініціалізації рядків:

– оголосити масив типу char, за допомогою оператора присвоювання = в подвійних лапках вказати необхідний текст, нуль-термінатор \0 додасться автоматично, розмір масиву при цьому вказувати не обов'язково (визначається компілятором): `char str[]="Hello world";` Використовуючи порожні лапки при ініціалізації, ми присвоюємо кожному елементу масиву значення \0. Таким чином рядок буде очищений від «сміття» інших програм;

– оголосити масив типу char, текст не присвоювати, при цьому вказати розмір масиву: `char str[20];`

Класичний спосіб оголошення масиву `char str[100] = {'H','e','l','l','o',' ','w','o','r','l','d','\0'};` при роботі з рядками не використовується.

Основні функції для роботи з рядками наведені у додатку Е. Для виведення рядка на екран достатньо звернутися до нього на ім'я: `cout<<str<<endl;` cout буде виводити на екран символ за символом, доки не зустріне в одному з елементів масиву символ кінця рядка \0, процес виводу перерветься. Таке звернення для звичайного символьного масиву (масиву без нуль-термінатора \0) неприпустимо. Також для виведення рядка на екран використовується функція puts().

Введення рядка з клавіатури можна здійснити за допомогою команди cin, проте з усього введеного зчитається послідовність символів до першого пробілу. Більш універсальною є функція gets(), яка зчитує всі введені символи з пробілами, поки не буде натиснута клавіша Enter.

Також для роботи з рядками використовуються вказівники. В цьому випадку до кожного рядка можна звернутися за допомогою вказівника на його перший символ. Більш детально робота із вказівниками буде описана у частині 2 цих методичних вказівок.

Приклад виконання роботи

Завдання 6.1. Визначити дію фрагмента програми

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

Розв’язання

Цей фрагмент програми виводить на екран число 2469

Завдання 6.2. Скласти програму, яка видаляє всі пробіли з рядка, та виконати її, якщо дано рядок "We study C++.".

1. Постановка задачі

Скласти програму, яка видаляє всі пробіли з рядка "We study C++.". Виконати програму.

2. Алгоритм розв’язання задачі

Алгоритм розв’язання задачі можна представити у вигляді такої послідовності дій:

Дія 1. Ініціалізувати рядок `str`.

Дія 2. Вивести рядок `str`.

Дія 3. Визначити *m* (довжину рядка `str`).

Дія 4. Повторювати *m* разів наступні дії (*i*= 0, 1, ..., *m*-1):

Дія 4.1. *j*-му символу рядка str присвоїти значення *i*-го символу рядка str;

Дія 4.2. Якщо *i*-й символ рядка str не пробіл, то збільшити значення лічильника *j* на 1;

Дія 4.3. Збільшити значення лічильника *i* на 1.

Дія 5. Якщо *j*<*i*, то *j*-му символу рядка str присвоїти значення кінця рядка `\0`.

Дія 6. Вивести рядок str після перетворення.

Запишемо алгоритм розв’язання задачі мовою C++, позначивши рядок через `str`, символи якого мають тип `char`. Змінні *i* та *m* позначимо відповідно як `i`, `m`, які мають тип `int`.

3. Текст програми

```
//програма видалення пробілів з рядка str
#include <iostream>
#include <cstring>
#include <windows.h>
using namespace std;
int main() {
```

```
SetConsoleOutputCP(1251);
SetConsoleCP(1251);
char str[]="We study C++.";
int m, i=0, j=0;
cout<<"Рядок: "<<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<<"Рядок після перетворення: "<<str<<endl;
return 0;}
```

4. Результати роботи програми

```
Рядок: We study C++.
Рядок після перетворення: WestudyC++.
```

Контрольні питання

1. Що використовується для роботи з рядками?
2. Що означає нуль-термінатор `\0`?
3. Які існують способи ініціалізації рядків?
4. Як можна вивести рядок на екран?
5. Як можна ввести рядок з клавіатури?
6. Наведіть основні функції роботи з рядками.

СПИСОК РЕКОМЕНДОВАНОЇ ЛІТЕРАТУРИ

1. **Павловская, Т.А.** C/C++. Программирование на языке высокого уровня [Текст] / Т.А. Павловская. – СПб. : Питер, 2006. – 461 с.

2. **Прата, С.** Язык программирования C++. Лекции и упражнения. – 6-е изд., обновл. и расшир. [Текст] / С. Прата. – М. : Диалектика – Вильямс, 2016. – 1248 с.

3. **Страуструп, Б.** Программирование: принципы и практика с использованием C++. – 2-е изд.; пер. с англ. [Текст] / Б. Страуструп. – М. : ООО «И.Д. Вильямс», 2016. – 1328 с.

4. **Страуструп, Б.** Язык программирования C++. Краткий курс. – 2-е издание [Текст] / Б. Страуструп. – К. : Диалектика, 2020. – 320 с.

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

6. **Шилдт, Г.** Самоучитель C++. – 3-е издание [Текст] : пер. с англ. / Г. Шилдт. – СПб. : BHV – Санкт-Петербург, 2003. – 688 с.

7. **Шилдт, Г.** C++: базовый курс. – 3-е издание [Текст] : пер. с англ. / Г. Шилдт. – К. : Диалектика, 2019. – 624 с.

Додаток А

Алфавіт мови С++

Алфавіт мови С++ складають:

– букви латинського алфавіту: `A – Z, a – z` та символ підкреслення `_` (ASCII-код 95);

– арабські цифри: `0 – 9`;

– спеціальні символи: `+ – * / % = < > . , : ; ’ ” ( ) [ ] { } # $ ^ ~ ! ? & |`;

– символи-розділителі: символ пропуску (ASCII-код 32) та управляючі символи (наведені у додатку Б);

– службові (зарезервовані) слова (наведені у додатку В).

Додаток Б

Управляючі символи мови С++

Послідовності символів, що починаються зі зворотної косої риски (\), називають керуючими, або escape-послідовностями – символи, які виштовхуються в потік виводу, з метою форматування виводу або друку деяких керуючих знаків С++. Основний список керуючих символів мови С++ представлений у таблиці Б.1.

Керуючі послідовності використовуються для опису певних спеціальних символів усередині строкових літералів і розглядаються як один символ.

Таблиця Б.1 – Управляючі символи мови С++

Керуюча послідовність	Опис
\'	одинарна лапка
\"	подвійна лапка
\?	літерал знаку питання
\\	зворотний слеш
\0	нульовий символ
\a	звуковий сигнал
\b	видалення попереднього символу
\f	нова сторінка
\n	новий рядок
\r	повернення каретки
\t	горизонтальна табуляція
\v	вертикальна табуляція
\nnn	символ ASCII у вісімковій нотації
\xnn	символ ASCII в шістнадцятковій нотації
\unnnn	символ юнікоду в шістнадцятковій нотації, якщо ця escape-послідовність використовується в багатобайтовій знаковій константі або строковому літералі юнікоду

Додаток В

Службові слова мови С++

Службові (зарезервовані) слова – це обмежена група слів, сенс яких зафіксовано в мові. Службові слова не можна вживати як ідентифікатори змінних, констант і 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	

Ідентифікатори `signed` та `volatile` зарезервовані для застосування в майбутньому.

Додаток Г

Базові математичні функції мови С++

Базові математичні функції мови програмування С++ містяться в заголовочному файлі `cmath.h` (аналог файлу `math.h` для мови С). Первісно в мовах С та С++ підтримувався один і той же набір з 22 математичних функцій. По мірі розвитку мови С++ до нього додалися перевантажені версії цих оригінальних функцій, які явно призначені для прийому значень типу `float` та `long double` (на відміну від т ипу `double` у мові С). Дії, що виконуються функціями, залишилися тими ж. Усі кути задаються в радіанах. У таблиці Г.1 наведено базові математичні функції мови С++.

Таблиця Г.1 – Базові математичні функції мови С++

Функція	Опис
`acos(a)`	арккосинус a, де -1.0 < a < 1.0
`asin(a)`	арксинус a, де -1.0 < a < 1.0
`atan(a)`	арктангенс a
`atan2(a, b)`	арктангенс b/a
`ceil(a)`	найменше ціле, яке перевищує або дорівнює a
`cos(a)`	косинус a
`cosh(a)`	гіперболічний косинус a
`exp(a)`	значення експоненти (e^a)
`fabs(a)`	абсолютне значення (модуль) a
`floor(a)`	найбільше ціле, яке менше або дорівнює a
`fmod(a, b)`	залишок від ділення a/b
`frexp(a, int *exp)`	розбиває число a на мантису і експоненту
`ldexp(a, int exp)`	повертає значення виразу $a*2^{exp}$
`log(a)`	натуральний логарифм a
`log10(a)`	десятковий логарифм a
`modf(a, *i)`	розбиває a на цілу і дробову частини
`pow(a, b)`	зведення a в ступінь b
`sin(a)`	синус a
`sinh(a)`	гіперболічний синус a
`sqrt(a)`	корінь квадратний з a, де a більше або дорівнює 0
`tan(a)`	тангенс a
`tanh(a)`	гіперболічний тангенс a

Додаток Д

Операції мови С++ за пріоритетом

В таблиці Д.1 наведені операції за пріоритетом з описом їх позначень, назв та синтаксису. Найвищий пріоритет мають операції у верхній частині таблиці.

Таблиця Д.1 – Операції мови С++ за пріоритетом

Позначення	Назва	Синтаксис
`::` `::`	Дозвіл області видимості Глобальне ім’я	*ім’я_класу* `::` *елемент* `::` *ім’я*

`.`
`->`
`[]`
`()`
	Вибір елемента через об’єкт
	Вибір елемента через вказівник
	Елемент масиву
	Виклик функції
		об’єкт `.` *елемент*
		вказівник `->` *елемент*
		вказівник `[`*вираз*`]`
		вираз `(`*аргументи*`)`

`()`
`sizeof`
`sizeof`
	Задання значення
	Розмір об’єкта
	Розмір типа
		тип `(`*аргументи*`)`
		`sizeof` *вираз*
		`sizeof (`*тип*`)`

`++`
`++`
`--`
`--`
`~`
`!`
`-`
`+`
`&`
`*`
`new`
`new[]`
`delete`
`delete[]`
`()`
	Постфіксний інкремент
	Префіксний інкремент
	Постфіксний декремент
	Префіксний декремент
	Доповнення
	Заперечення
	Унарний мінус
	Унарний плюс
	Отримання адреси
	Разіменування
	Створити
	Створити масив
	Вилучити
	Вилучити масив
	Приведення типу
		ідентифікатор`++`
		`++`*ідентифікатор*
		ідентифікатор`--`
		`--`*ідентифікатор*
		`~`*вираз*
		`!`*вираз*
		`-`*вираз*
		`+`*вираз*
		`&` *ідентифікатор*
		`*`*вираз*
		`new` *тип*
		`new` *тип*`[]`
		`delete` *вказівник*
		`delete[]` *вказівник*
		`(`*тип*`)` *вираз*

`.*` `->*`	Вибір вказівника через об’єкт Вибір вказівника через вказівник	*об’єкт* `.*` *вказівник-елемента* *вказівник*`->*`*вказівник-елемента*

`*`
`/`
`%`
	Множення
	Ділення
	Остача від ділення
		вираз `*` *вираз*
		вираз`/`*вираз*
		вираз `%` *вираз*

`+` `-`	Додавання Вилучення	*вираз* `+` *вираз* *вираз* `-` *вираз*
`<<`	Зсув вліво	*вираз* `<<` *вираз*

>>	Зсув вправо	*вираз* >> *вираз*
< <= > >=	Менше ніж Менше ніж або дорівнює Більше ніж Більше ніж або дорівнює	*вираз* < *вираз* *вираз* <= *вираз* *вираз* > *вираз* *вираз* >= *вираз*
== !=	Дорівнює Не дорівнює	*вираз* == *вираз* *вираз* != *вираз*
&	Побітне І	*вираз* & *вираз*
^	Побітне вилучаюче АБО	*вираз* ^ *вираз*
|	Побітне АБО	*вираз* | *вираз*
&&	Логічне І	*вираз* && *вираз*
||	Логічне АБО	*вираз* || *вираз*
?:	Умовний вираз	*вираз* ? *вираз* : *вираз*
= *= /= %= += -= <<= >>= &= |= ^=	Просте присвоєння Множення та присвоєння Ділення та присвоєння Остача від ділення та присвоєння Додавання та присвоєння Вилучення та присвоєння Зсув вліво та присвоєння Зсув вправо та присвоєння Побітне І та присвоєння Побітне АБО та присвоєння Побітне вилучающе АБО та присвоєння	*ідентифікатор* = *вираз* *ідентифікатор* *= *вираз* *ідентифікатор* /= *вираз* *ідентифікатор* %= *вираз* *ідентифікатор* += *вираз* *ідентифікатор* -= *вираз* *ідентифікатор* <<= *вираз* *ідентифікатор* >>= *вираз* *ідентифікатор* &= *вираз* *ідентифікатор* |= *вираз* *ідентифікатор* ^= *вираз*
,	Кома	*вираз*, *вираз*

Додаток Е

Основні функції для роботи з рядками мови С++

Основні функції для роботи з рядками мови програмування С++ містяться у декількох заголовочних файлах: `cstring` (аналог файлу `string.h` для мови С), наведені у таблиці Е.1; `cstdio` (аналог файлу `stdio.h` для мови С), наведені у таблиці Е.2; `cstdlib` (аналог файлу `stdlib.h` для мови С), наведені у таблиці Е.3.

Таблиця Е.1 – Основні функції для роботи з рядками файлу `cstring`

Функція	Опис
`int strlen(char* str)`	підраховує довжину рядка (кількість символів без урахування `\0`)
`char *strcat(char *dest, const char *scr)`	приєднує рядок `src` в кінець рядка `dest`, одержаний рядок повертається в якості результату
`char *strncat(char *dest, const char *scr, int n)`	приєднує `n` символів рядка `src` в кінець рядка `dest` і повертає рядок `dest`
`char *strcpy(char *dest, const char *scr)`	виконує копіювання рядка `src` в рядок `dest` і повертає рядок `dest`
`char *strncpy(char *dest, const char *scr, int n)`	виконує копіювання n символів рядка `src` в рядок `dest` і повертає рядок `dest`
`int ctrcmp(const char *s1, const char *s2)`	порівнює два рядки в лексикографічному порядку з урахуванням відмінності великих і малих літер; функція повертає `0`, якщо рядки збігаються, повертає `-1`, якщо `s1` розташовується в упорядкованому за алфавітом порядку раніше, ніж `s2`, і `1` - в протилежному випадку
`int ctrncmp(const char *s1, const char *s2, int n)`	порівнює n символів двох рядків в лексикографічному порядку з урахуванням відмінності великих і малих літер; функція повертає `0`, якщо рядки збігаються, повертає `−1`, якщо `s1` розташовується в упорядкованому за алфавітом порядку раніше, ніж `s2`, і `1` - в протилежному випадку
`getline(char *s, int n)`	призначена для введення з клавіатури рядка `s` з пробілами, в рядку не повинно бути більше `n` символів
`char *strchr(const char *str, int c);`	повертає вказівник на перше входження символу c в рядок, на який вказує `str`, якщо символ c не знайдений, повертає `NULL`

`char *strupr(char *str)`	перетворює символи рядка, на який вказує `str`, в символи верхнього регістру, після чого повертає його
`char *strlwr(char *str)`	перетворює символи рядка, на який вказує `str`, в символи нижнього регістра, після чого повертає його

Таблиця Е.2 – Основні функції для роботи з рядками файлу `cstdio`

Функція	Опис
`int sprintf(char*, const char*, ...)`	вивід результату в рядок
`int sscanf(const char*, const char*, ...)`	введення значень з рядка
`char* gets(char*)`	зчитує потік символів зі стандартного пристрою вводу в рядок до тих пір, поки не буде натиснута клавіша ENTER
`int puts(const char*)`	вивід рядка на екран з переводом курсору на наступний рядок

Таблиця Е.3 – Функції перетворення для роботи з рядками файлу `cstdlib`

Функція	Опис
`int atoi(const char *str)`	перетворює рядок у ціле число, в разі невдалого перетворення повертається `0`
`long atol(const char *str)`	перетворює рядок у довге ціле число, в разі невдалого перетворення повертається `0`
`double atof(const char *str)`	перетворює рядок у дійсне число, в разі невдалого перетворення повертається число `0`

Додаток Ж

Використання маніпуляторів в мові С ++

Система введення/виведення мови С++ включає спосіб зміни параметрів форматування потоку. Для цього використовуються спеціальні функції, що називаються маніпуляторами (manipulators), які можуть включатися в вирази введення/виведення. Опис стандартних маніпуляторів знаходиться в заголовочному файлі `iomanip.h` та приведено в таблиці Ж.1. Для використання маніпуляторів з параметрами в програму необхідно включити заголовочний файл `iomanip.h`.

Таблиця Ж.1 – Стандартні маніпулятори мови С++

Маніпулятор	Опис	Вико-ристання
`dec`	Введення/виведення даних в десятковій формі	введення та виведення
`endl`	Виведення символу нового рядка з передаванням в потік всіх даних з буфера	виведення
`ends`	Виведення нульового символу	виведення
`flush`	Передача в потік вмісту буфера	виведення
`hex`	Введення/виведення даних в шістнадцятковій системі	введення та виведення
`oct`	Введення/виведення даних в вісімковій формі	введення та виведення
`resetiosflags(long f)`	Скидає прапори, зазначені в f	введення та виведення
`setbase(int base)`	Встановлює базу числення, що дорівнює параметру `base`	виведення
`setfill(int ch)`	Встановлює символ заповнення рівним `ch`	виведення
`setiosflags(long f)`	Встановлює прапори, зазначені в `f`	введення та виведення
`setprecision(int p)`	Встановлює кількість цифр після коми	виведення
`setw(int w)`	Встановлює ширину поля рівної `w`	виведення
`ws`	Пропускає початковий символ-роздільник	введення

Маніпулятори можуть використовуватися в складі виразів введення/виведення. Нижче наведено приклад програми, в якій демонструються маніпулятори для зміни формату виведення. Слід звернути увагу, як маніпулятори з'являються в послідовності операторів

введення/виведення. Коли маніпулятори не мають аргументів, як, наприклад, маніпулятор `endl`, за ними не йдуть дужки. Причина цього в тому, що оператору << передається адреса маніпулятора.

```
#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 роботи програми:

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

Додаток И

Оцінка похибки при роботі з дійсними числами на цифровому комп'ютері

Похибка при роботі з дійсними числами на цифровому комп'ютері може складатися з трьох складових:

– похибка подання;

– похибка накопичення;

– похибка чисельного методу, що використовується.

Необхідно мати можливість оцінювати цю похибку, тобто оцінювати, з якою точністю буде представлений результат, а також зменшувати її.

Для зменшення похибки подання необхідно використовувати дійсні типи даних, які визначаються більшою кількістю байт, а для зменшення похибки накопичення – спеціальні прийоми програмування, які її мінімізують.

Проілюструємо ці компоненти похибки на прикладі табулювання функції $y=x^3$ при зміні значення x від $a=-1$ до $b=1$ з кроком $h=0{,}1$.

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

Результат роботи програми:

```
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=1.0000001192 y=1.0000003576 delta y=0.0000003576
```

У наведеному прикладі на початку програми (до циклу) ілюструється похибка подання: `float h=0.1` при форматному виведенні на екран `setprecision(10)` надає `h=0.1000000015`. В даному випадку похибка подання складає `0.0000000015` або $1.5*10^{-9}$.

У прикладі 1 звернемо увагу на значення змінної x, що обчислюється в середині циклу за формулою $x_i=x_{i-1}+h$. Такий спосіб обчислення призводить до збільшення похибки накопичення з кожною ітерацією циклу. Крім того, через отриману похибку накопичення, значення функції y при x=1 в циклі одержати не вдалося (це значення пораховане окремо після циклу).

Похибку накопичення можливо зменшити, якщо значення змінної x в циклі обчислювати за наступною формулою $x_i=a+i*h$, що наведено в прикладі 2.

Приклад 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 роботи програми:

```
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=0.00000000000000056 y=0.00000000000000000 delta
y=0.00000000000000000
x=0.10000000000000060 y=0.00100000000000002 delta
y=0.00000000000000002
x=0.20000000000000070 y=0.00800000000000009 delta
y=0.00000000000000007
x=0.30000000000000040 y=0.02700000000000010 delta
y=0.00000000000000010
x=0.40000000000000080 y=0.06400000000000029 delta
y=0.00000000000000014
x=0.50000000000000110 y=0.12500000000000080 delta
y=0.00000000000000083
x=0.60000000000000090 y=0.21600000000000080 delta
y=0.00000000000000083
x=0.70000000000000070 y=0.34300000000000080 delta
y=0.00000000000000167
x=0.80000000000000040 y=0.51200000000000120 delta
y=0.00000000000000000
x=0.90000000000000130 y=0.72900000000000310 delta
y=0.00000000000000222
x=1.00000000000000000 y=1.00000000000000000 delta
y=0.00000000000000000
```

На початку прикладу 2 за рахунок використання типу даних `double` (розмір – 8 байт) замість `float` (розмір – 4 байти) вдалося зменшити похибку подання: `double h=0.1` при форматному виведенні на екран `setprecision(18)` дає `h=0.100000000000000010`. В даному випадку похибка подання становить `0.000000000000000010` або $1.0*10^{-17}$.

Як видно з результатів роботи програми (приклад 2), вдалося також зменшити похибку накопичення. У третьому стовпці наведено значення абсолютної похибки $\Delta y=|y_{\text{точне}}-y_{\text{наближене}}|$. У прикладі 1 Δy (`delta y`) збільшується і суттєво більше, ніж в прикладі 2.

Оцінка та зменшення похибки ряду чисельних методів буде розглянута в дисципліні «Чисельні методи».

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

Коректор *О. Є. Вакула*
Комп'ютерне верстання *В. В. Москаленко*

Формат 60×84/16. Ум. друк. арк. 3,5. Тираж 100 прим. Вид. № 23. Зам. № 1101-04.
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 № 6402 від 19.09.2018 р.