**ЛАБОРАТОРНА РОБОТА № 2.**

**Основні поняття мови програмування С++.**

**Оператори розгалуження програми у мові С++**

# **2.1. Мета роботи**

Вивчити основні поняття мови програмування С++, операції, стандартні функції, оператори розгалуження програм.

**2.2. Теоретичні відомості**

2.2.1. Мова С++

# Алфавіт мови C++ включає:

Програма в мові C++ записується символами базового алфавіту (*basic character set*), який містить:

* великі (A–Z) і малі (a–z) літери латинського алфавіту та символ підкреслення ( \_ );
* арабські цифри від 0 до 9, шістнадцяткові цифри від A до F;
* символи арифметичних операцій (+, –, \*, /, %, ++, —);
* символи побітових операцій (<<, >>, &, |, ~, ^);
* символи логічних відношень (<, <=, ==, !=, >, >=);
* символи логічних операцій (&&, ||, !);
* розділові знаки (, ; : пропуск);
* спеціальні знаки (., =, –>, ?, \, $, #, ‘, ʺ);
* символи дужок (, ), [, ], {, }.

Із символів алфавіту формуються базові елементи мови або лексеми (*tokens*), ними можуть бути:

* ідентифікатори (*identifiers*);
* ключові слова (*key words*);
* оператори (*operators*);
* константи (*constants*);
* знаки пунктуації (*punctuators*).

Межі лексем визначаються іншими лексемами-розділювачами або операторами.

# Ключові слова мови С++

Ключовими (службовими, зарезервованими) словами називають ряд зарезервованих ідентифікаторів, які вживаються для формування конструкцій мови і мають фіксоване значення. За змістовим навантаженням службові слова поділяються на такі основні групи:

* специфікатори типів – char, int, long, typedef, short, float, double, enum, struct, union, signed, unsigned, void;
* кваліфікатори типів – const і volatile;
* класи пам’яті – auto, extern, register, static;
* для формування настанов – for, while, do, if, else, switch, case, continue, goto, break, return, default, sizeof.

У табл. 2.1 подано перелік основних ключових слів C++.

Таблиця 2.1

Ключові слова C++

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| asm | delete | goto | register | throw |
| auto | do | if | return | try |
| break | double | inline | short | typedef |
| case | else | int | signed | typename |
| catch | enum | long | sizeof | union |
| char | explicit | new | static | unsigned |
| class | extern | operator | struct | virtual |
| const | float | private | switch | void |
| continue | for | protected | template | volatile |
| default | friend | public | this | while |

Як розділювачі лексем застосовуються такі символи: пропуск, табуляція, символ нового рядка, коментар. Між довільними двома лексемами допускається довільна кількість символів-розділювачів. Крім того, деякі лексеми (ʺ\*ʺ,ʺ+ʺ,ʺ,ʺ,ʺʺ, ʺ(ʺ, ʺ–>ʺ тощо) самі є розділювачами і відділяти їх від інших лексем символами-розділювачами необов’язково.

# Ідентифікатори мови С++

Ідентифікатори використовуються для визначення змінних, функцій, структур, об'єднань, їх полів, перечислень, власних типів даних, міток, назв макросів та їх параметрів. Один і той ж ідентифікатор може визначати різні програмні об'єкти у різних ділянках програми в залежності від області його видимості (*scope of identifier visibility*). Правила запису ідентифікаторів:

1. Ідентифікатори складаються з букв, цифр і символу підкреслення (“**\_**”) (до складу ідентифікатора не може входити будь-який спеціальний символ).
2. Першим символом повинна бути буква або символ підкреслення. Однак, не рекомендується починати ідентифікатори із символу підкреслення оскільки багато змінних стандартних бібліотечних програм починається саме з цього символу.
3. Враховується регістр. Не можна плутати в ідентифікаторах великі і малі букви (**Х** і **x** – це два різні ідентифікатори).
4. Немає обмежень на максимальну довжину ідентифікатора (однак, це залежить від конкретного компілятора і згідно стандарту максимальна довжина повинна становити хоча б 31 символ).

Константи мови С++

Константами (*constant*) називають незмінні значення. Розрізняють цілі, дійсні, символьні і рядкові константи. Компілятор, відзначивши константу у якості лексеми, відносить її до одного з типів за зовнішнім поданням. Константи також називають літералами (*literals*).

Характерна особливість їх визначень порівняно з визначеннями змінних – наявність специфікатора (*qualifier*) константи const, наприклад:

const double pi=3.14159F;

Спосіб визначення кожної константи залежить від її типу. Константи мови С++ слід поділяти на *літеральні* та *типізовані*.

Літеральна константа – це лексема, яка є зображенням фіксованого числового, рядкового або символьного значення. Такі константи бувають цілі, дійсні, символьні та рядкові (*табл. 2.2*).

Цілі константи можуть бути десятковими, вісімковими та шiстнадцятковими.

Довгі цілі константи (long) мають літеру l або L в кінці, наприклад: 32768L; 0777777l; 0XFL. Для задання константи, без знака (unsigned) застосовується літера u (U), наприклад, 65535u. Довгі константи без знака записуються з використанням двох літер відразу: (ul, UL) або (lu, LU).

*Таблиця 2.2*

Літеральні константи мови С++

|  |  |  |
| --- | --- | --- |
| Константа | Формат | Приклади |
| Цiла | Десятковий: послідовність десяткових цифр (0, 1, 2, 3, 4, 5, 6, 7, 8, 9), що починається не з нуля, якщо це не число нуль | 9, 0, 217925 |
|  | Вісімковий: нуль, за яким розташовані вісімкові цифри (0, 1, 2, 3, 4, 5, 6, 7) | 02, 050, 07245 |
|  | Шістнадцятковий: 0х або 0Х, за яким розташовані шістнадцяткові цифри  (0, 2, 3, 4, 5, 6, 7, 8, 9, А, В, С, D, Е, F) | 0x1B9, 0X00FF |
| Дійсна | Десятковий:[цифри].[цифри]  Експоненційний: [цифри][.][цифри]{Е|е}[+|-][цифри] | 9.7, .001, 87. 0.7Е6, .15е-3 9.2, 920 е-2, 92.Е-1, .92Е1 |
| Символьна | Один або два символи, що подаються в апострофах | ′А′ , ′ю′ , ′\* ′ , ′\0′, ′\n′, ′\012′,  ′\х07\х07′ |
| Рядкова | Послідовність символів, що подаються в лапках | ʺRESULTʺ,  ʺ\t sum\_\_s=\0x5\nʺ |

Дійсні числа у мовах програмування мають дві форми подання: десяткову (природну) та експоненційну (показникову).

*Десяткова форма* дійсного числа – це звичайний десятковий формат запису дійсного числа, ціла частина дійсного числа відділяється від дробової розділювачем – крапкою, а не комою, наприклад 10.123, 1.0123, 1012.3, 0.0010123.

Експоненційна форма дійсного числа використовується для запису дуже великих або дуже малих чисел, для яких задавати зайві нулі не зовсім зручно, наприклад, 1 .0123\*1020,1.0123\*10-10. У цій формі запису числа можна виокремити такi основні характеристики: знак числа, мантису числа, знак порядку та порядок числа. Відзначені характеристики дійсного числа зберігаються у пам’яті комп’ютера. Число у показниковій формі може бути подано, наприклад, так: 1.0123Е-10. Мантиса записується ліворуч від знаку експоненти **(Е або е**), порядок – праворуч. Символ **Е (е)** означає основу степеня 10, і компілятор розпізнає цей запис як форму подання дійсного числа. Символ пропуску всередині числа не допускається, а для відділення цілої частини від дробової використовується не кома, а крапка. При додатних значеннях числа і мантиси знак ʺ+ ʺ можна не вказувати.

Як десяткова, так і екcпоненційна форми запису допускають відсутність або цілої частини, або дробової, але не двох одразу.

За замовчуванням, всі дійсні константи мають тип **double** – подвійну точність, що найчастіше займає у пам’яті 64 біти, тобтo 8 байтів. Якщо програміста не влаштовує тип за замовчуванням, його можна вказати явно за допомогою спеціальних літер. Так, додавши літеру **f або F**, константі надають дійсний тип **float** зі звичайною точністю, наприклад, 8.5f. Якщо в поданні константи використовується літера **L або l**, тоді вона має тип **long double**.

Зображення від’ємної цілої або дійсної константи вважається константним виразом, що складається зі знаку унарної операціi зміни знаку (-) та константи, наприклад: -273, -2730.е-1, -273L.

Символьні константи мають один або два символи, що подаються в апострофах. Односимвольні константи займають у пам’яті один байт і мають стандартний тип **char** (***character ‒* символ**). Двосимвольні константи займають два байти і мають тип **int**. Символьні константи мають цілий тип і їх можна використовувати як цілочислові операнди у виразах.

Заслуговують уваги послідовності, що починаються зі знаку ʺ\ʺ, їх називають ***керуючими*** або ***escape*-послідовностями.** Символ зворотної скісної риски ʺ\ʺ (*backslash*) використовується для запису кодів, що не мають графічного зображення, для запису символiв, а також для виведення символьних констант, якщо їх коди заданi у вісімковому та шістнадцятковому вигляді (табл. 2.3).

Символьні константи зображують значення типу char, які займають один байт. Цей байт як зображення числа без знаку дає код символу (число від 0 до 255), а зі знаком – від -128 до 127.

Таблиця 2.3.

**Керуючі послідовності мови С++**

|  |  |
| --- | --- |
| \а | звуковий сигнал |
| \b | повернення на крок |
| \f | переведення сторінки (формату) |
| \n | новий рядок |
| \r | повернення каретки |
| \t | горизонтальна табуляція |
| \v | вертикальна табуляція |
| \\ | символ \ – зворотна коса риска |
| \’ | символ '– апостроф |
| \” | символ ʺ – лапки |
| \0 | нуль-символ |
| \? | знак запитання |
| \0ddd | вісімковий код символу |
| \0xddd | шістнадцятковий код символу |

Відповідність між числами від 0 до 127 і символами зафіксовано в *Американському стандартному коді для обміну інформацією* (так звана *таблиця ASCII*), решті кодів можуть відповідати різні набори символів. Наприклад, константа ' ' (пробіл) має код 32, а '\0' – код 0. Окремими властивостями таблиці ASCII є такі:

* символам '0', '1', …, '9' відповідають послідовні коди від 48 до 57;
* символам 'A', 'B', …, 'Z' – від 65 до 90;
* символам 'a', 'b', …, 'z' – від 97 до 122.

Настанова cout<<'Z'; виводить значення константи 'Z' – на екрані з'являється Z. Аналогічно можна вивести інші символьні константи. Для того, щоб наступне повідомлення виводилося з нового рядка, можна скористатися настановами cout<<endl; або cout<<'\n';.

Настановою

cout<<*константа*;

обробляється не *константа*, а зображене нею значення. Послідовність символів, утворена за значенням, може відрізнятися від константи.

Приклад 2.1:

#include <iostream>

using namespace std;

int main() {

cout<<'1';

cout<<'\n';

cout<<'Y';

cout<<'e';

cout<<'s';

cout<<endl;

system("pause");

return 0;

}

Рядкова константа (рядковий літерал) – це послідовність cимволів, яка подається в лапках (тобто в символах ʺ) і зберігається у неперервній ділянці пам’яті, наприклад: ʺЦе рядковий літералʺ.

Рядки, які записані у програмі поспіль або через символи пропуску, при компілюванні конкатенуються (склеюються).

Довгу рядкову константу можна розмістити також у декількох рядках. У цьому випадку ставиться зворотна коса риска і на`тискається клавіша *Enter*. Наприклад:

ʺПрограма виконує те, \

Що Ви їй вказали виконувати, але не те, \

що Ви хотіли, щоб вона виконувала. ʺ

Використовуючи рядкові константи можна записати так:

#include <iostream>

using namespace std;

int main() {

cout<<ʺПрограма виконує те,\n що Ви їй вказали виконувати, але не те,\n що Ви хотіли, щоб вона виконувала. ʺ;

system("pause");

return 0;

}

Змінні

У програмуванні змінна є моделлю (зображенням) об'єкта в пам'яті комп'ютера. ***Змінна*** – це іменована область пам’яті, у якій зберігаються дані визначеного типу. Змінна має ім’я, розмір та інші атрибути, такі як область видимості, час існування тощо. Ім’я змінної служить для звертання до області пам’яті, у якій зберігається її значення. Тип змінної та її ім'я задаються настановою оголошення імені змінної, що має вигляд:

**<ім'я\_типу> <ім'я\_змінної>.**

Після імені типу можна записати кілька імен змінних відділяючи їх комами. оголошення імені змінної одночасно є означенням змінної, тобто виділення ділянки пам'яті, на яку вказує ім'я змінної. У програмному коді змінна позначається ідентифікатором (символічним іменем). За ідентифікатором здійснюється звернення до значення змінної.

Базові (фундаментальні) типи даних

Для опису основних типів мови C++ використовують такі службові слова:

* int (цілий);
* char (символьний);
* wchar\_t (розширений символьний)
* bool (логічний);
* float (дійсний);
* double (дійсний з подвійною точністю);
* void (без типу).

Почнемо з типу void, який лише формально віднесено до базових типів. Множина значень цього типу є порожньою, не має значень Він використовується для визначення функцій, які не повертають значення, для визначення порожнього переліку аргументів функції, як базовий тип для вказівників.

Усі подані фундаментальні типи, окрім void, є арифметичними в тому розумінні, що для кожного з них задано звичайні арифметичні операції. Логічний, символьний і цілі типи відносяться до так званих інтегральних типів (*integral types*). Над ними, крім арифметичних, можна виконувати також логічні операції. Крім інтегральних типів, якими кодуються цілі числа, існують типи для зображення дійсних чисел. Це так звані дійсні типи з плаваючою крапкою (*floating poin*t): float, double і long double, які відрізняються одне від одного точністю.

Типи **int, char, bool** називають ***цілими***, а типи **float** та **double** – дійсними з плаваючою крапкою. Код, який формує компілятор для оброблення цілих значень, відрізняється від коду для значень з плаваючою крапкою.

Для уточнення внутрішнього подання та діапазону значень базових типів мова C++ використовує чотири специфікатори типу:

* short (короткий);
* long (довгий);
* signed (знаковий);
* unsigned (беззнаковий).

У табл. 2.4 подано діапазони значень та розміри базових типів даних (для 16-розрядного і 32-розрядного процесорів). Розмір однакового типу даних може відрізнятися на комп’ютерах різних платформ, а також може залежати від використаної операційної системи. Тому, при оголошенні тієї або іншої змінної потрібно чітко уявляти, скільки байт вона буде займати в пам’яті комп’ютера, щоб запобігти проблемам, які пов’язані з переповненням і неправильною інтерпретацією даних. Діапазони кожного з типів повинні бути перевірені для конкретного комп’ютера.

Таблиця 2.4

Базові типи даних

|  |  |  |
| --- | --- | --- |
| Тип | Розмір, байт | Значення |
| bool | 1 | true або false |
| unsigned short int | 2 | від 0 до 65 535 |
| short int | 2 | від -32 768 до 32 767 |
| unsigned long int | 4 | від 0 до 4 294 967 295 |
| long int | 4 | від -2 147 483 648 до 2 147 483 647 |
| int (16 розрядів) | 2 | від -32 768 до 32 767 |
| int (32 розряди) | 4 | від -2 147 483 648 до 2 147 483 647 |
| unsigned int (16 розрядів) | 2 | від 0 до 65 535 |
| unsigned int (32 розряди) | 4 | від 0 до 4 294 967 295 |
| char | 1 | від 0 до 256 |
| float | 4 | від 1.2е-38 до 3.4е38 |
| double | 8 | від 2.2е-308 до 1.8е308 |
| long double | 10 | від 3.4е-4932 до 3.4е+4932 |

2.2.2. Структура програми

Формально програмний код мовою С++ складається з таких основних частин типової структури:

* директиви препроцесорного оброблення;
* оголошення (опис зовнішніх змінних та функцій);
* функції програми;
* головна функція – програми main().

Отримання виконуваного коду програми мовою С++ здійснюється в декілька етапів: препроцесорне оброблення тексту програми, компілювання, редагування зв’язків (компонування) та виконання. Препроцесорне оброблення програми здійснюється за допомогою спеціальної програми-препроцесора CPP.EXE перед компілюванням програми.

Задачею препроцесора (препроцесор – це складова частина компілятора (програма), яка проводить попереднє оброблення програми) є перетворення початкового тексту програми. На виході препроцесора отримується змінений текст програми мовою С++. Попереднє оброблення проводиться перед процесом компілювання, а спеціальні функції попередньо обробляються. У результаті отримується розширена програма C ++, а потім вона передається компілятору.

Препроцесор одні дії виконує за замовчуванням, інші дії вказуються за допомогою спеціальних директив у тексті програми.

Директиви препроцесора (preprocessor directive) – це команди препроцесора, вони завжди починаються знаком #. Наприклад, #include <ім'я файлу> вказує препроцесору, що вміст заданого файлу необхідно обробити так, якби воно знаходилося у початковій програмі в тій точці, в якій знаходиться ця директива.

Оголошення є записом, який містить опис деяких об'єктів. У мові С++ оголошення є різновидом настанов і повинні завершуватися символом ʺ ; ʺ. Оголошення, так як і настанови, виконуються в процесі роботи програми і результатом виконання оголошення, у більшості випадків, є створення імен об'єктів та самих об'єктів з відповідними властивостями.

Оголошення, яке не є частиною тіла довільної функції, є глобальним оголошенням. Об'єкт, створений таким оголошенням також називається глобальним, він може бути як простим, так і динамічним. Простором імен, створених глобальними оголошеннями, є програма в цілому.

Одна з функцій повинна мати ім’я main(). Її ім'я main фіксоване в усіх програмних кодах мовою С++ і завжди записується однаково. Виконання програмного коду починається з першої настанови цієї функції.

У загальному випадку програма складається з декількох функцій, які не перетинаються (тобто ʺвкладенняʺ однієї функції в іншу неприпустиме). Перед функціями і між ними можуть бути записані оголошення об’єктів даних і настанови препроцесорного оброблення. Функції користувача, які викликаються у головній функції main(), слід обов’язково описати до їх використання.

# Операції

Умовно операції в мові C++ можна розбити на такі групи:

1. Арифметичні операції:
   1. унарні: **+**;**-**;
   2. бінарні **+**;**-**;**\***;**/**; (додавання, віднімання, множення, ділення).

До операції ділення в мові C++ потрібно відноситись дуже уважно. Якщо обидва операнди цілого типу, то і результат буде цілого типу. Наприклад **S=2/5;** – в результаті виконання цього оператора, змінній **S** присвоїться значення **0**, щоб одержати правильний результат необхідно щоб хоча б один операнд був дійсного типу, тобто **S = 2.0/5.0**;

У мові C++ є ще одна бінарна операція **%** - знаходження залишку від ділення цілих чисел. Наприклад **K=7%2;** – присвоїти змінній цілого типу **K** значення **1**, оскільки **7:2=3** і **1** в залишку. До змінних дійсного типу ця операція не застосовується.

1. Операції порівняння: **>**; **>=**; **<**; **<=**;

|  |  |  |
| --- | --- | --- |
| 3) Операційні рівності: | **==** – рівне; | **!=** – не рівне |
| 4) Логічні операції: | **!** – логічне “ні”; | **||** – логічне “або”; |

**&&** – логічне “і”.

5) Інкрементні та декрементні операції.

Інкрементна операція **++** додає 1 до свого операнда.

Оператор **n++;** можна записати **n=n+1**;

Декриментна операція **--** віднімає 1 від свого операнда.

Розрізняють два види цих операцій:

1. префіксні **++n** – змінна n збільшується на 1 до того, як використовується у виразі;
2. постфіксні **n++** – змінна n збільшується на 1 після того, як її значення буде використано у виразі.
3. Побітові операції

В мові C++ існує 6 операцій для роботи з бітами:

|  |  |
| --- | --- |
| **&** | – побітове “і”; |
| **|** | – побітове “або”; |
| **^** | – побітове “виключне “або” (XOR); |
| **~** | – побітове “ні”; |
| **>>** | – зсув вправо; |
| **<<** | – зсув вліво. |

**& –** *побітове “і”* бінарна операція, що по розрядах порівнює два двійкові числа. Результат дорівнює 1, якщо обидва операнди рівні 1 у цьому розряді, тобто:

**1 0 0 1 0 0 1 1**

**&**

**0 0 1 1 1 1 0 1**

**0 0 0 1 0 0 0 1**

**| –** *побітове “або”***.** Результат 1, якщо хоча б у одного операнду у цьому розряді 1, тобто:

**1 0 0 1 0 0 1 1**

**|**

**0 0 1 1 1 1 0 1**

**1 0 1 1 1 1 1 1**

**^ –** *побітове “виключне “або”.*Для кожного розряду результат дорівнює 1, якщо один з двох відповідних розрядів дорівнює 1, але не обидва одночасно:

**1 0 0 1 0 0 1 1**

**^**

**0 0 1 1 1 1 0 1**

**1 0 1 0 1 1 1 0**

**~ –** *побітове “ні”.*Унарна операція, яка заміняє кожну 1 на 0, а 0 на 1:

|  |  |  |
| --- | --- | --- |
|  | **~0b10010011 == 0b01101100** |  |

**>> –** *зсув вправо*. Зсуває розряди лівого, операнд вправо на кількість позицій вказаних у правому операнді:

|  |  |  |
| --- | --- | --- |
| **0b10010011 << 2** | **==** | **0b00100100** |
| **<< - зсув вліво -** вказані в правому операнді: |  | зсуває розряди лівого операнда вліво на кількість розрядів, що |
| **0b10010011 >> 2** | **==** | **0b01001100** |

позиції, які звільняються заповнюються нулями.

1. Тернарний оператор

Умовний оператор **if** у мові C можна замінити операцією виду “**?**”**:**”

**z=(a<b)?a:b;**

Цей оператор відповідає оператору умовного переходу такого виду:

**if(a<b)**

**z=a;**

**else**

**z=b;**

1. Операція присвоєння.

Операція присвоювання може мати такий вигляд: **<змінна> = <вираз>;**

**<змінна> <знак операції>= <вираз>;** наприклад:

**S=S+4;**

**S+=4;**

В операції присвоювання можуть використовуватись такі операції:

**+**,**-**,**\***,**/**,**%**,**<<**,**>>**,**&**,**|**

Введення/виведення даних

Обов’язковим етапом рзв’язання навіть найпростішої задачі на комп’ютері не обходиться без введення початкових даних та виведення проміжних і остаточних результатів. Введення даних – це передавання інформації ззовні в оперативну пам’ять (із зовнішнього носія); виведення даних – зворотний процес, коли дані після оброблення передаються з оперативної пам’яті на зовнішній носій. Зовнішнім носієм може слугувати дисплей, друкуючий пристрій, жорсткий диск, флеш-пам’ять тощо. Передавання даних у програму та виведення результатів виконання програми є необхідним елементом програми.

Алгоритмічні мови програмування використовують концепцію поелементного введення/виведення даних. Виконуючи введення даних з клавіатури, комп’ютер тимчасово зупиняє виконання програмного коду і очікує на введення значення для змінної. У відповідь слід з клавіатури набрати деяку послідовність символів, яка зображує значення (ці символи візуалізуються на екрані). Введені символи запам’ятовуються у буфері та передаються функціям введення тільки після натиснення клавіші Enter.

Буфер – це область пам’яті для тимчасового зберігання даних. Максимальний обсяг буфера становить 128 символів (байтів). Завдяки наявності буфера можливе редагування даних під час їх введення.

Організація введення та виведення даних у різних мовах програмування виконується по різному: або відповідними настановами, або за допомогою стандартних підпрограм. Так, у мові С++ немає вмонтованих засобів введення/виведення даних. Для організації відповідних дій використовуються стандартні бібліотечні функції. Бібліотеки С++ підтримують два основних способи введення/виведення:

потокове введення/виведення;

форматоване введення/виведення.

Поняття потоку

У мові С++ основним поняттям введення/виведення даних є потік – послідовність символів або інших даних. У програмі *потік є представником фізичного файлу* на зовнішньому носії даних (диску, клавіатурі або екрані монітора), а операції обміну даними із файлом зображено як операції відбору даних із потоку або дописування (долучення) їх до нього.

У програмуванні існує поняття стандартних файлів введення/виведення – зазвичай ними є клавіатура та екран. У С++-програмі їм відповідають стандартний потік введення з ім'ям cin і стандартний потік виведення з ім'ям cout. Отже, імена cin і cout, означені у файлі iostream, насправді позначають не клавіатуру та екран, а потоки, які відповідають цим пристроям у програмному коді. На початку виконання програмного коду обидва потоки cin і cout є порожні.

Із засобів оброблення потоків розглянемо лише оператори введення або вставляння (*insertion*) даних у потік >> і виведення <<, або відбору даних з потоку.

Є два способи надати змінній значення – ввести із зовнішнього носія або присвоїти. У найпростішому випадку настанова введення значення змінної має формат:

**cin >> ім'я*\_*змінної;**

і реалізовує введення даних з клавіатури. Нагадаємо, ім'я cin, як і cout, оголошені у файлі iostream.

У настанові введення можна записати кілька імен змінних, кожне після відповідного оператора >>. За виконання такої настанови треба набрати з клавіатури відповідну кількість початкових значень, відокремивши їх одним або кількома порожніми символами. Приклади:

1. Наприклад, є змінна char c; і виконується настанова

cin >> c;.

Щоб присвоїти змінній значення 'A', треба натиснути послідовно на клавіші A та Enter. Якщо перед клавішею A кілька разів натиснути на клавіші Enter або Space, тоді результат буде такий самий.

2. Наприклад, змінні

double a, b, c;

є коефіцієнтами квадратного рівняння. Надати їм дійсні значення можна за допомогою трьох окремих настанов:

cin >> a;

cin >> b;

cin >> c;

Те ж саме буде задано і однією настановою:

cin >> a >> b >> c;

В обох ситуаціях з клавіатури слід набрати три значення, натискаючи між ними на клавішу пробілу, табуляції або Enter.

Виконання настанов закінчиться, коли після третьої константи буде натиснуто на Enter (перелік введення буде вичерпано).

Практично перед кожною настановою введення з клавіатури варто записати настанову виведення, яка запрошує до введення значень і вказує, скільки та яких типів.

Приклад. Задання значень дійсним змінним, які відповідають значенням коефіцієнтів квадратного рівняння, можна оформити таким чином:

cout << "Ведіть значення коефіцієнтів квадратного рівняння\n";

cout << "(Три значення дійсного або цілого типу):\n";

cin >> a >> b >> c;

Після появи цього запрошення курсор буде переведено до початку наступного рядка екрану і з нього почнеться відтворення символів, набраних з клавіатури.

*Вивести значення змінної* на екран можна за допомогою настанови відбору з потоку виведення, яка має формат:

**cout << ім'я\_змінної;**

Настанова

**cout << вираз;**

обчислює та виводить значення виразу.

Наприклад, фрагмент програмного коду:

cout<< "Введіть число: ";

cin>> x;

cout<< "Квадрат цього числа: " << x\*х << endl;

Приклад. Після введення значень трьох дійсних змінних за настановою

cin >> a >> b >> c;

можна вивести їх на екран, відокремивши пробілами:

cout << a;

cout << ' ';

cout << b;

cout << ' ';

cout << c;

Ті ж дії можна виконати й однією настановою:

cout << a << ' ' << b << ' ' << c;

Якщо введено значення 1, –3, 2, тоді буде надруковано 1 –3 2.

Пропонуємо Вам з’ясувати, що буде виведено на екран, якщо під час виконання поданого програмного коду введено символи 15 і n та натиснуто на клавішу Enter?

#include <iostream>

using namespace std;

int main()

{

int n; char a;

cin >> n >> a;

cout<<"n="<<n<<ʹ ʹ<<"a="<<a<<endl;

system("pause");

return 0;

}

Зауважимо, що при виведенні тексту для коректного візуалізування літер кирилиці слід застосувати команди:

setlocale(0,".1251");

або

setlocale(LC\_ALL, "Ukrainian");

Виведення замість літер кирилиці усілякої ʺабабагаламагиʺ спричинено тим, що, наприклад, Visual Studio в консольних застосунках використовує для набраного тексту кодування Windows 1251, а для введеного тексту – кодування DOS. Коректно візуалізувати введений за допомогою cin>> текст дозволить команда

setlocale(LC\_ALL, ".OCP");

повернувши початкові налаштування кодування.

Крім того, встановити потрібну кодову таблицю для потоків введення/виведення можна і такими способами:

system("chcp 1251");

system("chcp 1251 > null").

або

#include <Windows.h>

………………………………………………………

SetConsoleCP (1251);

SetConsoleOutputCP(1251);

Після цього у властивостях консольного вікна на вкладці Шрифт слід вибрати Lucida Console.

Доволі зручною є можливість виведення за допомогою cout<< чисел не лише в десятковому форматі, а і у шістнадцятковому або вісімковому, використовуючи модифікатори dec, hex і oct усередині потоку виведення, наприклад:

#include <iostream>

using namespace std;

int main ()

{

setlocale(LC\_ALL, "Ukrainian");

cout<< "Вісімковий:\t\t " << oct << 10 << " " << 255 << endl;

cout<< "Шістнадцятковий: \t "<< hex<<10<<" "<<255<<endl;

cout<< "Десятковий:\t\t " << dec << 10 << " " << 255 << endl;

system("pause");

return 0;

}

Результатом виконання цих команд будуть рядки:

Вісімковий: 12 377

Шістнадцятковий: a ff

Десятковий: 10 255

Варто відзначити, що використання одного з цих модифікаторів залишиться коректним, доки програма не завершиться, або не буде використано інший модифікатор.

# Стандартні математичні функції бібліотеки **<cmath>**

|  |  |  |  |
| --- | --- | --- | --- |
| Ім’я функції | Математичний запис | Тип і межі зміни  аргументів | Тип результату |
| **sin(x)** | sin x | **double** | **double** |
| **cos(x)** | cos x | **double** | **double** |
| **tan(x)** | tg x | **double** | **double** |
| **asin(x)** | arcsin x | **double** x[-1,1] | [-/2, /2] |
| **acos(x)** | arccos x | **double** x [-1,1] | [0, ] |
| **atan(x)** | arctg x | **double** x [- /2, /2] | **double** |
| **sinh(x)** | sh x | **double** | **double** |
| **cosh(x)** | ch x | **double** | **double** |
| **tanh(x)** | th x | **double** | **double** |
| **exp(x)** | ex | **double** | **double** |
| **log(x)** | ln x | x>0 | **double** |
| **log10(x)** | lg x | x>0 | **double** |
| **pow(x,y)** | xy | **double** | **double** |
| **sqrt(x)** | √x | x>0 | **double** |
| **fabs(x)** | |x| | **double** | **double** |
| **ldexp(x,n)** | x · 2n | x-**double**, n-**int** | **double** |
| **fmod(x,y)** | Залишок від ділення дійсних чисел х на у | **double** | **double** |

2.2.3. Перетворення типів

В операторах і виразах бажано використовувати змінні і константи однакового типу. Якщо у виразі є змішування типів компілятор автоматично перетворить типи за такими правилами. Якщо операція виконується над змінними різних типів, то обидві змінні переводяться до “вищого” з двох типів.

Порядок типів від “вищого” до “нижчого” має такий вигляд:

1. **double**; 2) **float**; 3) **long**; 4) **int**; 5) **short**; 6) **char**.

В операторах присвоювання результат перетворюється до типу змінної, якій присвоюється це значення. Це може бути як “підвищення” типу так і “пониження”, наприклад в програмі:

Найкраще при написанні програми уникати перетворення типів, особливо в бік “пониження”. Можливий варіант явного приведення типів

**(<iмя типу>)<змінна або константа>.**

Так у прикладі:

**m = 1.6 + 1.8;**

**n = (int)1.6 + (int)1.8;**

змінній **m** присвоїться значення **3**, оскільки спочатку додадуться два дійсних числа **1.6** і **1.8** від результату **3.4** відкинеться дробова частина. Змінній **n** присвоїться значення **2**, оскільки **(int)1.6==1**, **(int)1.8==1**; **1+1==2**.

2.2.4. Оператори

Будь-яка програма складається з послідовності операторів. Ознакою закінчення оператора є крапка з комою “**;**”. Так запис **S=5** не є оператором, це просто вираз, а **S=5;** це вже оператор присвоювання.

Блок – це група операторів, що міститься у фігурних дужках, вони використовуються:

* щоб згрупувати кілька логічно зв’язаних операторів в один;
* як тіло функції;
* для локалізації дії описів (визначають область видимості ідентифікаторів).

# Оператор **if**

Оператор **if-else** використовується для вибору одного з двох варіантів рішення. Синтаксичний опис оператора **if-else**:

**if(<вираз>)**

**<оператор 1>;**

**else**

**<оператор 2>;**

Обчислюється “**вираз**”, якщо його значення “істина” (тобто не нуль) виконується “**оператор 1**”, якщо “не істина” (тобто нуль) виконується “**оператор 2**”. Частина **else** може бути відсутня. При вкладених **if-else** необхідно пам’ятати, що **else** відноситься до внутрішнього **if**:

**if(x>0)**

**if(a>b)**

**z = a;**

**else**

**z = b;**

Якщо треба змінити порядок необхідно використати фігурні дужки, тобто виділити блок:

|  |
| --- |
| **if(x>0)**  **{**  **if(a>b) z = a;**  **}**  **else z = b;** |

Щоб розгалузити програму можна використати конструкцію: **else if**:

**if(<вираз 1>)**

**<оператор 1>; else if(<вираз 2>)**

**<оператор 2>;**

**else if(<вираз 3>)**

**<оператор 3>;**

**else**

**<оператор 4>;**

Якщо “**вираз 1**” – “істина” виконується “**оператор 1**”, якщо ні – перевіряється “**вираз 2**”. Якщо “**вираз 2**” – “істина”, виконується “**оператор 2**” і так далі.

# Оператор **switch**

Оператор **switch** (перемикач) використовується для вибору одного з багатьох варіантів.

Синтаксис оператора **switch**:

|  |
| --- |
| **switch(<вираз>)**  **{**  **case <константа вибору 1>:<оператор 1>;**  **case <константа вибору 2>:<оператор 2>;**  **default: <оператор n>**    **}** |

Якщо “вираз” співпадає з одною із констант вибору, то виконується відповідний оператор або блок операторів. Якщо “вираз” не співпадає з жодною з констант вибору – виконується оператор після слова “**default**”.

“Вираз” і константи вибору повинні бути цілого типу, або типу **char**. Заборонено використовувати в якості константи вибору змінну. Оператор **break** здійснює негайний вихід з оператора **switch**. Якщо цього оператора немає, то будуть виконані оператори всіх варіантів після вибраного.

## 2.3. Контрольні запитання

1. Правила запису ідентифікаторів.
2. Які Ви знаєте типи даних і як вони описуються?
3. Структура програми на мові C.
4. Що таке побітові операції?
5. Що таке оператор, ознака закінчення оператора.
6. Як здійснюється узгодження типів у виразах?
7. Яка різниця між оператором і блоком у мові C?
8. Синтаксис умовного оператора **if**.
9. Напишіть приклад оператора **switch**.
10. Для чого використовується оператор **break**?

## 2.4. Лабораторне завдання

1. Вивчити основні поняття мови програмування С, операції, стандартні функції, оператори розгалуження програм.
2. Одержати індивідуальне завдання.
3. Побудувати блок-схеми алгоритмів відповідно до завдання.
4. Скласти програми на алгоритмічній мові C згідно завдання.
5. Відлагодити програми, виконати обчислення, проаналізувати отримані результати.

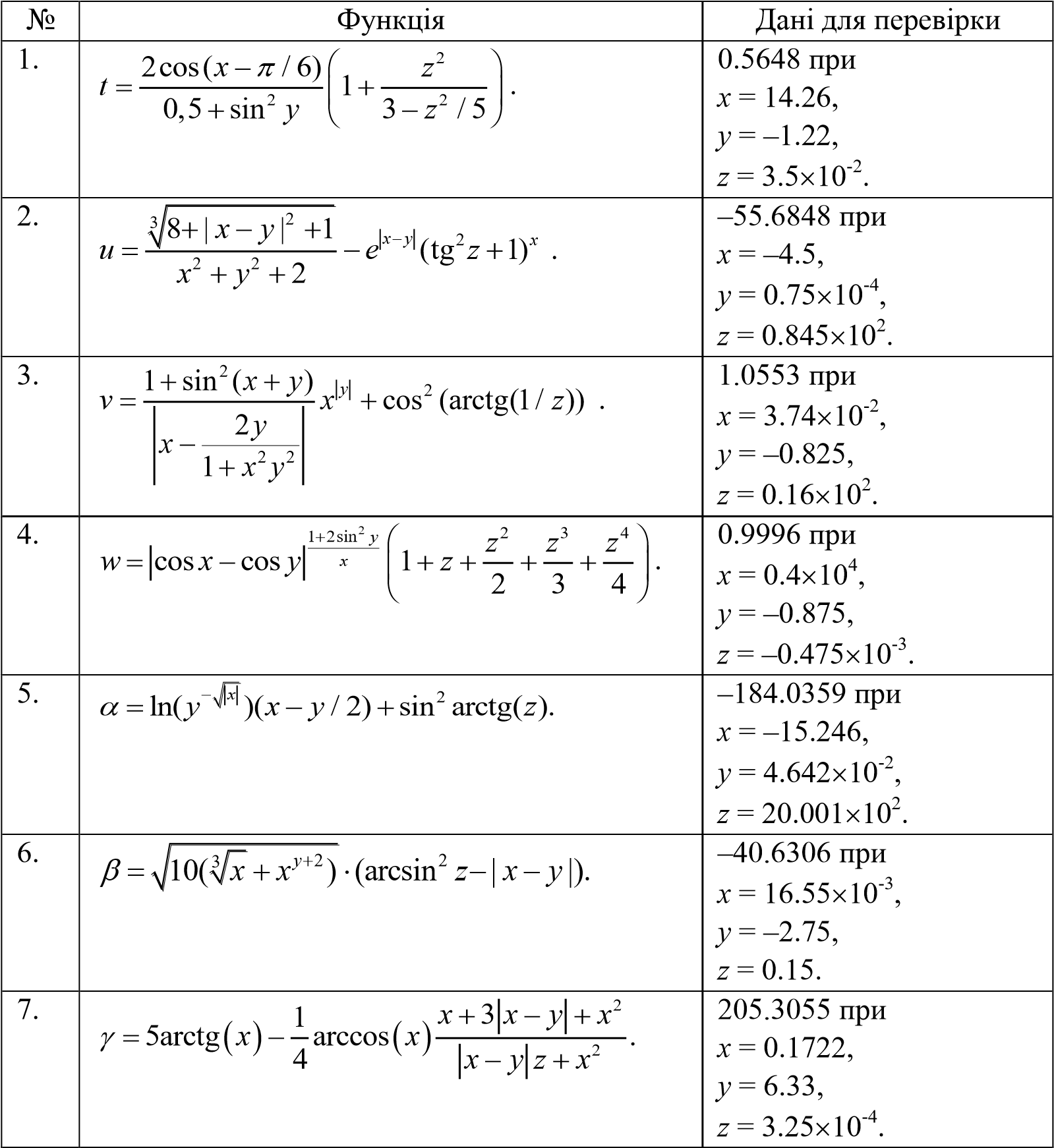
**2.5. Зміст звіту** 1. Титульний аркуш.

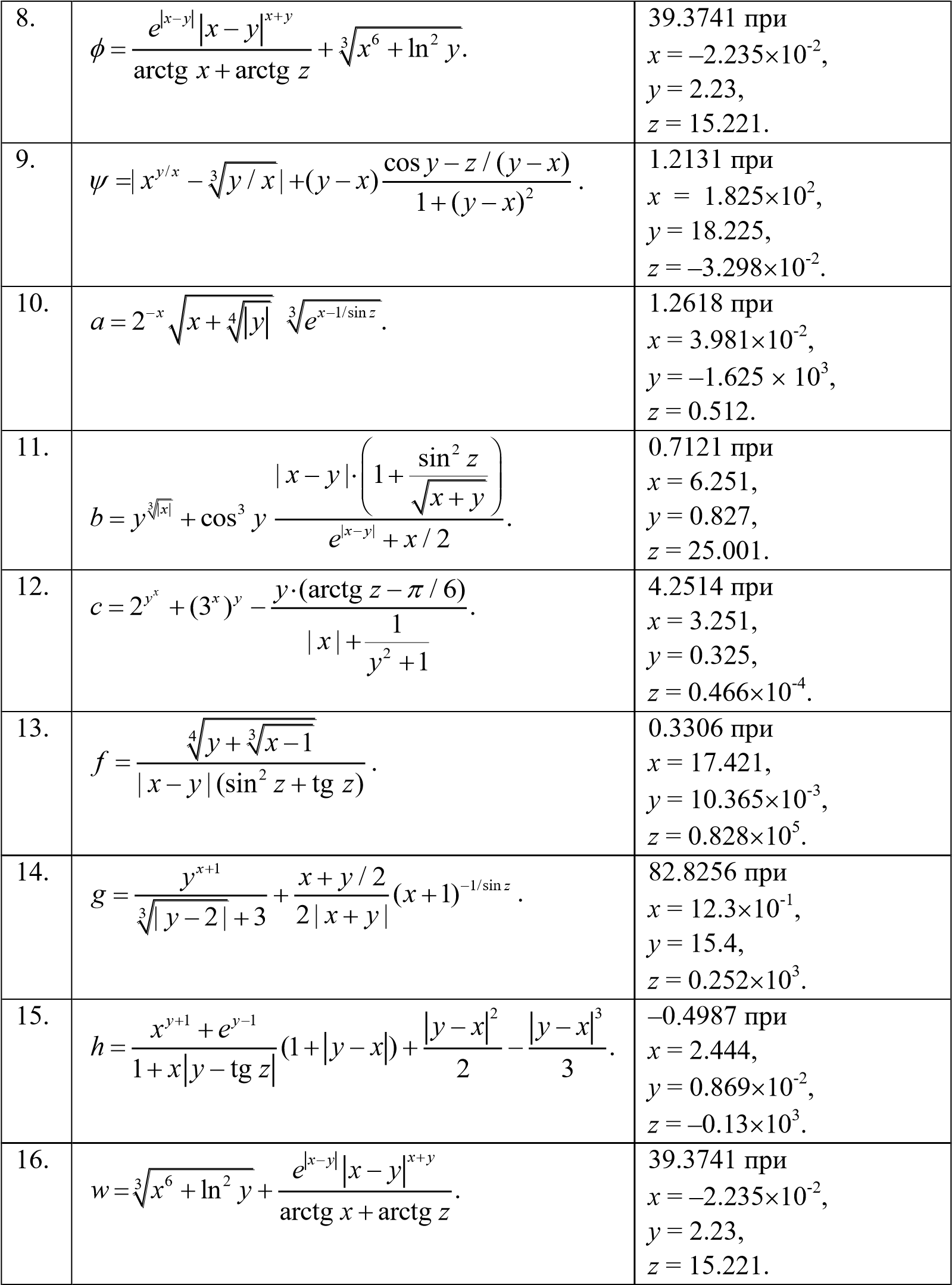
1. Мета роботи.
2. Індивідуальне завдання.
3. Блок-схема алгоритмів у відповідності до завдання.
4. Тексти програм у відповідності до завдання.
5. Результати обчислень.
6. Аналіз результатів, висновки.

**2.6. Індивідуальні завдання**

# Завдання 1

Скласти програму обчислення функцій для різних значень її аргументів. Аргументи вводити з клавіатури. Вивести на екран значення функції. Передбачити у програмі обхід алгебраїчних операцій, які можуть при певних значеннях аргументів мати невизначений результат, тобто ділення на нуль, добування кореня парного степеня з від’ємного числа, логарифма від’ємного числа і тому подібне, при цьому вивести на екран повідомлення про те, що функція не визначена.





# Завдання 2

Скласти програму обчислення функцій для різних значень її аргументів. Аргументи вводити з клавіатури. Вивести на екран значення функції та проміжної змінної.

