КОМУНАЛЬНИЙ ЗАКЛАД «КІРОВОГРАДСЬКИЙ ОБЛАСНИЙ ІНСТИТУТ
ПІСЛЯДИПЛОМНОЇ ПЕДАГОГІЧНОЇ ОСВІТИ ІМЕНІ ВАСИЛЯ СУХОМЛИНСЬКОГО»

C++ в задачах. Олімпійські кроки

Навчальний посібник

Друкується за рішенням вченої ради комунального закладу
«Кіровоградський обласний інститут післядипломної педагогічної
освіти імені Василя Сухомлинського»
(від 10.06.2025 року, протокол № 3)

Кропивницький
2025

УДК 004.43:373.5.016(075)
К 27

Карявка С.С., Паламарюк М.Р. С++ в задачах. Олімпійські кроки : навчальний посібник. Кропивницький: КЗ «КОІППО імені Василя Сухомлинського», 2025. 116 с.

Посібник «С++ в задачах. Олімпійські кроки» орієнтований на учнів і педагогів, які готуються до олімпіад з інформатики. Матеріал охоплює основи програмування мовою С++ — від базових конструкцій до обчислювальних та не обчислювальних алгоритмів — та подається через велику кількість оригінальних практичних завдань із поступовим ускладненням.

Автори акцентують увагу на розвитку алгоритмічного мислення і вмінні самостійно аналізувати задачі. Посібник стане цінним ресурсом для тренування перед змаганнями та для глибшого засвоєння стратегічних тем шкільного курсу інформатики.

Рецензенти:

Фурсикова Тетяна Володимирівна – доцент кафедри інформаційно-комунікаційних технологій та безпечного освітнього середовища КЗ «КОІППО імені Василя Сухомлинського», доктор педагогічних наук;

Громко Григорій Юрійович – вчитель інформатики комунального закладу «Нечаївський ліцей ім. Ю. І. Яновського» Компаніївської селищної радиКропивницького району Кіровоградської області.

Відповідальний за випуск – Віталій ДМИТРУК

Зміст

Передмова

Перші кроки у вивченні програмування в школі часто супроводжуються постійним пошуком і підбором матеріалу, роботою з великими обсягами інформації. Найважливіше на цьому етапі — виконання завдань, які мають бути цікавими, практичними та зрозумілими.

Основна мета цієї роботи — охопити ключові теми процедурного програмування та надати матеріал для роздумів: запропонувати серію завдань від простих до ускладнених логічними головоломками. За основу для розв'язку завдань обрано мову програмування C++, хоча деякі способи вирішення будуть продубльовані іншими сучасними мовами програмування, щоб можна було порівняти їхні особливості та спільні елементи коду.

На початку кожної теми наведено коротку довідкову інформацію про відповідну керівну конструкцію та приклад її використання мовою C++, щоб читач міг швидко повторити матеріал і перейти до аналізу програмного коду. Для глибшого розуміння теми або розділу варто самостійно опрацьовувати додаткову літературу, якої сьогодні не бракує. Головне — вміти шукати та аналізувати інформацію.

Ця збірка завдань буде корисною як для педагогів та учнів, що готуються до олімпіад, так і для тих, хто прагне попрактикуватися в написанні коду. Адже непросто одразу взятися за створення складних, великих програм або працювати над масштабними проєктами без попереднього досвіду написання невеликих завершених програм.

Завдання підібрані так, щоб охопити більшість необхідних тем, які вчать продуктивно використовувати синтаксис та можливості мови програмування, включати всі стратегічно важливі конструкції. Це також допоможе розвинути алгоритмічне мислення, яке є безмежним для дослідження.

Деякі завдання подаються у жартівливій формі або з хитромудрим описом. До частини із них додано коментарі, що допоможуть більш продуктивно засвоїти матеріал і запам’ятати потрібні конструкції.

Ця збірка — своєрідна шпаргалка для учня, який робить свої перші кроки в програмуванні, обравши C++ як інструмент.

Вступ

Коли ви вперше стикаєтеся з програмуванням, це може здатися складною та заплутаною справою. Безліч нових термінів, синтаксичних правил та логічних концепцій — все це може викликати відчуття, що це не для вас. Але насправді, як і будь-яка інша навичка, програмування вимагає лише часу, практики та правильного підходу. У посібнику кожен знайде дієві поради, як зробити перші кроки максимально простими та зрозумілими. Завдяки рекомендаціям поступово і впевнено опануєте основи C++ , а навчання буде зрозумілим та цікавим.

Мова програмування C++ була обрана не випадково. Це одна з найпопулярніших і найпотужніших мов, яку використовують як для створення простих програм, так і для розробки масштабних програмних систем. Однак, як і будь-яка інша мова, C++ має свої особливості.

Коротка історія C++

C++ народилася з мови програмування C, яка на початку 1970-х років стала справжнім проривом для створення високопродуктивних програм. У 1983 році Б'ярн Страуструп додав до C елементи об'єктно-орієнтованого програмування, створивши те, що ми зараз знаємо як C++. Відтоді мова зазнала численних оновлень, що дозволяє їй залишається актуальною та популярною серед розробників.

C++ стала потужним інструментом для реалізації справді революційних проєктів у світі IT.

Серед успішних рішень застосування технологій, бібліотек та алгоритмів реалізованих цією потужною мовою є всесвітньо відомі проєкти

Значна частина операційної системи Windows, якою користуються мільйони користувачів, включаючи ядро та інші критичні компоненти, була написана на C++.

Основа браузера Google Chrome потребувала високої продуктивності та взаємодії між HTML, CSS, JavaScript та інших технологій, саме тому реалізована засобами цієї мови.

Серед популярних систем, які використовують принцип відкритого коду, є популярна база даних **MySQL**.

Свій вклад C++ внесла в роботу таких знаменитих сервісів як: Adobe Photoshop, AutoCAD, серверна частина алгоритмів для YouTube, Amazon, Facebook, Bing, Spotify, Blender, Skype та багато інших.

Навіщо вивчати C++

C++ відкриває величезні можливості для тих, хто прагне розібратися в тонкощах програмування на низькому рівні та отримати контроль над усіма аспектами програми. Цю мову використовують у багатьох сферах, включаючи розробку ігор, драйверів та вбудованих систем, системного програмування. Освоївши C++, ви отримаєте міцну базу для вивчення інших мов програмування та зможете вирішувати завдання різного рівня складності.

Як користуватися цією книгою

Книга побудована так, щоб зробити ваше навчання максимально ефективним. Кожна тема представлена чітко і послідовно: невелика теоретична основа та практичні завдання. Окрім прикладів коду, ви знайдете пояснення кожного кроку, щоб легко зрозуміти новий матеріал.

Рекомендуємо виконувати всі запропоновані завдання, щоб досягти найкращих результатів. Спершу ознайомтеся з прикладами, спробуйте написати свій код і перевірте його роботу. Якщо щось здається складним, повертайтеся до матеріалу та спробуйте ще раз. Практика — найкращий шлях до майстерності!

У книзі також наведені коментарі, підказки та поради, що допоможуть уникнути типових помилок і краще зрозуміти матеріал. Якщо ж якесь питання залишиться не до кінця зрозумілим, не соромтеся звертатися до додаткових джерел інформації або обговорити це з іншими програмістами.

Вперед до вивчення C++!

1. Структура програми, написаної мовою C++

Програма, написана мовою C++, складається з кількох основних компонентів.

Заголовкові файли (header files): це файли з розширенням .h або .hpp, які містять оголошення функцій, класів, констант і змінних. Наприклад, заголовковий файл #include <iostream> містить оголошення для вводу та виводу.

Функція main(): це головна функція, з якої починається виконання програми. Без цієї функції програма не буде виконуватися.

```
1. int main() {
2.   \\ код програми
3.   return 0;
4. }
```

Оголошення змінних: у C++ всі змінні повинні бути оголошені перед використанням. Наприклад:

```
1. int a = 55;
2. double b = 3.14;
```

Програма може складатися з кількох функцій, які виконують певні завдання. Наприклад:

```
1. void printHello() {
2.  std::cout << "Hello, world!";
3. }
```

У програмі завжди присутні оператори, які виконують обчислення (арифметичні, логічні, умовні оператори тощо).

Цикли і умовні конструкції: Цикли (for, while) та умовні оператори (if, switch) використовуються для контролю потоку виконання програми.

Класи та об'єкти: Якщо програма використовує об'єктно-орієнтоване програмування, вона може містити класи та об'єкти для створення більш складної структури даних.

```
1. class Car {
2. public:
3.     void drive() {
4.         std::cout << "Driving!";
5.     }
6. };
```

Примітка:

В C++ namespase – це механізм, який використовується для організації коду та запобігання конфліктам між іменами функцій, змінних, класів та інших об'єктів. Простір імен допомагає структурувати великий код та уникати помилок, які виникають через використання однакових імен в різних частинах програми.

Наприклад:

```
1. namespace myNamespace {
2.     int x = 100;
3.
4.     void myFunction() {
5.         std::cout << "Hello from myNamespace!" << std::endl;
6.     }
7. }
```

Щоб звернутися до об'єктів (змінних, функцій, класів) у просторі імен, потрібно використовувати його ім'я разом з оператором :: (оператор області видимості)

```
1. int main() {
2.     // Використовуємо змінну та функцію з myNamespace
3.     std::cout << myNamespace::x << std::endl;   // Виведе: 10
4.     myNamespace::myFunction();   // Виведе: Hello from
myNamespace!
5.     return 0;
6. }
```

Одним з найбільш відомих і часто використовуваних просторів імен у C++ є **std**. У ньому знаходяться всі стандартні функції та об'єкти мови C++ (наприклад, std::cout, std::string, std::vector).

Щоб кожного разу не писати std::, можна використовувати директиву using:

```
1. using namespace std;
2.
3. int main() {
4.     cout << "Hello, world!" << endl;  // Тепер можна
використовувати без std::
5.      return 0;
6. }
```

Класичним на початку є наведення першої програми, яка виводить на екран вітання "**Hello world!**".

Це приклад простого шаблону, з якого має починатись більшість ваших програм.

```
1. #include <iostream> // підключення бібліотек
2.
3. using namespace std; //оголошення простору імен
4.
5. int main() // підключення головної функції
6. {
7.     cout << "Hello world!" << endl; //виведення
8.     return 0;
9. }
```

Код варто писати, щоб він був читабельним.

Під час написання перших програм слідкувати за читабельністю коду, вміти коментувати свої рішення, щоб вони стали доступними для сприйняття іншими.

Іноді над проєктами працює багато програмістів і в такому випадку варто розуміти, що є певні стандарти і правила, яких варто дотримуватись.

Використовуйте **зрозумілі** імена, що пояснюють суть змінної або функції.

Уникайте коротких і неінформативних назв, наприклад, x, y, якщо вони не мають очевидного значення.

Назви повинні бути **послідовними**. У різних частинах програми повинні використовуватися одні й ті самі терміни.

Додавайте до програм **коментарі** - короткі тексти в програмі, які слугують поясненням алгоритму або окремих фрагментів. У C++ прийнято використовувати коментарі двох видів:

// однорядковий, короткий коментар.

/* **Коментар** */ - це багаторядковий коментар, який може займати декілька рядків.

Варто уникати коментарів, які дублюють код.

Використовуйте **однакові відступи** по всьому проєкту. Стандартом є 4 пробіли або 1 табуляція. Не змішуйте пробіли з табуляціями в одному файлі.

Ліміт ширини рядка: зазвичай рекомендується обмежувати довжину рядка до 80 або 120 символів.

Завжди ставте **фігурні дужки** навіть для однорядкових блоків if, for, while, щоб уникнути помилок.

Розбивайте програму на **логічні модулі** і файли. Не перевантажуйте один файл занадто великою кількістю функцій або класів.

Функції повинні виконувати **одну задачу**. Якщо функція занадто велика або має кілька завдань, її слід розбити на менші функції.

Оголошувати змінні варто якомога ближче до місця їх використання.

Тестуйте код після кожної важливої зміни. Не відкладайте тестування до кінця розробки.

Варто слідкувати за оновленнями мови, адже розробники завжди намагаються спростити, оптимізувати і покращити роботу з кодом, і це є цілком класичний процес в програмуванні.

2. Псевдокод

Під час розв’язання завдань у деяких прикладах буде використано псевдокод, як спосіб пояснення до завдань.

Псевдокод — це наближений до людської мови опис алгоритму або програми, який не використовує жорсткий синтаксис конкретної мови програмування.

Він стає в нагоді, коли розробникам потрібно планувати, організовувати та описувати логіку роботи програми до написання коду на певній мові програмування.

Псевдокод не має жорстких правил і використовує зрозумілі конструкції для спрощення планування.

Є чудовим інструментом для планування написання коду певною мовою програмування.

Наприклад:

потрібно обчислити суму натуральних чисел від 1 до 10.

Загальний план реалізації:

- Ініціалізація змінної для суми.
- Перебір чисел від 1 до 10.
- Додавання кожного числа до суми.
- Виведення результату.

У вигляді псевдокоду задача матиме наступний вигляд:

```
1. ПОЧАТИ
2. Ініціалізувати змінну sum в 0
3. Для кожного числа від 1 до 10
4.     Додати число до sum
5. ВИВЕСТИ sum
6. КІНЕЦЬ
```

Для опису логіки програми в псевдокоді використовують конструкції типу: **ЯКЩО, ТО, ІНАКШЕ.**

Наприклад:

```
1. ЯКЩО число більше за нуль
2.     Вивести "Це число більше за нуль"
3. ІНАКШЕ
4.     ВИВЕСТИ "Це число менше за нуль"
```

Для опису конструкцій, які описують циклічні дії використовують конструкції:

- **ПОКИ НЕ:**
- **ДЛЯ КОЖНОГО ЧИСЛА:**
- **ДОКИ ДІЙСНО ЩО:**
- **ВИКОНУВАТИ ДО:**

У випадку, коли маємо складений алгоритм, і він передбачає окремі логічні блоки, варто використовувати підпрограми.

```
1. ФУНКЦІЯ ОБЧИСЛЕННЯ СУМИ
2.     ДЛЯ КОЖНОГО ЧИСЛА від 1 до 10
3.         ДОДАТИ число до суми
4.     ВИВЕСТИ суму
5. КІНЕЦЬ ФУНКЦІЯ
```

Доречним у псевдокоді також буде використання коментарів, як однорядкових так і багаторядкових.

// - однорядковий коментар, як і на мові програмування

/* **… коментар…** */ - багаторядковий коментар.

Завжди слід надати увагу ієрархії блоків. Контролювати відповідно відступи, щоб було зрозуміло, який блок включає в себе наступний, чи до якого блоку входить поточний.

Наприклад:

Потрібно знайти суму всіх парних чисел від 1 до 100.

Псевдокод матиме наступний вигляд і відразу стає зрозумілим, який приклад є вдалим.

Псевдокод без відступів	Псевдокод з ієрархією блоків
1. ПОЧАТИ 2. ВСТАНОВИТИ змінну sum в 0 3. ДЛЯ КОЖНОГО ЧИСЛА від 1 до 100 4. ЯКЩО число парне 5. ДОДАТИ число до sum 6. ВИВЕСТИ sum 7. КІНЕЦЬ	1. ПОЧАТИ 2. ВСТАНОВИТИ змінну sum в 0 3. ДЛЯ КОЖНОГО ЧИСЛА від 1 до 100 4. ЯКЩО число парне 5. ДОДАТИ число до sum 6. ВИВЕСТИ sum 7. КІНЕЦЬ

Псевдокод надає можливість створити алгоритм рішення без прив'язки до конкретної мови програмування.

Як приклад до попереднього завдання, може бути наступний код програми, реалізованою засобами C++:

```
 1. #include <iostream>
 2.
 3. int main() {
 4.     int sum = 0;
 5.     for (int i = 1; i <= 100; i++) {
 6.         if (i % 2 == 0) { // перевірка на те чи число парне
 7.             sum += i;
 8.         }
 9.     }
10.     std::cout << "Сума парних чисел: " << sum << std::endl;
11.     return 0;
12. }
```

2. Лінійні алгоритми. Змінні. Прості операції.

Величини, з якими працюють програмісти в C++, діляться на змінні та константи, а також можуть зберігатись як в пам’яті, так і в регістрах. Це так званні своєрідні комірки для зберігання даних.

Величина, значення якої протягом періоду виконання програми не змінюється, називається константою.

Ефективне використання змінних дозволяє економно використовувати пам’ять та зменшує час виконання програми, що в спортивному програмуванні є цінним.

Змінна завжди має ім’я та тип.

Ініціалізація — це процес присвоєння початкового значення змінній під час її оголошення. Якщо змінну не ініціалізувати, вона матиме випадкове значення, яке знаходиться в пам'яті.

```
1. int age = 25;
```

Після початкової ініціалізації в процесі роботи програми змінна може змінювати своє значення потрібну кількість разів, а константа значення не змінює протягом роботи всієї програми.

Тип змінної визначає, які дані може зберігати змінна, і скільки пам'яті їй буде виділено.

Змінні можна оголошувати окремо кожну і списками, у будь-якому місці програми, за необхідністю.

```
1. //оголошення змінних різного типу
2. int res, sum;
3. int a,b,c,p; float s;
4. long long x, res2;
5. float a,b,c,d;
6. float x,y=2.6;
7. char w=’a’;
8. char name[50];
```

```
 9. const int BITS_IN_WORD = 32;
10.
11. //приклади оголошення сталих
12. const vik=34, rist=180;
13. const float q = 5.56;
14. const float pi = 3.1415;//оголосити константу pi типу float
із значенням 3.1415
```

У мові C++ є стандартні константи, наприклад, число пі, (Pi) для цього слід підключити відповідну бібліотеку **math.h,** яка містить у собі відповідну інформацію.

Базові типи даних, які використовуються в програмах можна об'єднати в декілька груп.

Цілочисельні типи:

- int: ціле число (зазвичай 4 байти);
- short: коротке ціле число (зазвичай 2 байти);
- long: довге ціле число (зазвичай 4 або 8 байтів);
- long long: дуже велике ціле число (зазвичай 8 байтів).

Наприклад:

```
1. int age = 13;
2. long distance = 10000000;
```

Числа з плаваючою крапкою:

- float: число з плаваючою крапкою одинарної точності (4 байти);
- double: число з плаваючою крапкою подвійної точності (8 байтів).

Наприклад:

```
1. float temperature = 36.6f;
2. double pi = 3.1415926535
```

Символьні типи:

- char: зберігає символ або маленьке ціле число (1 байт).

Наприклад:

```
1. char grade = 'B'
```

Логічний тип:

- bool: зберігає логічне значення — true або false.

Рядкові типи:

- std::string: рядки символів, які представляють текст. Для роботи з рядками потрібне підключення бібліотеки <string>

Наприклад:

```
1. #include <string>
2. std::string name = "John Deere";
```

Область видимості — це частина програми, в межах якої можна використовувати змінну. Є кілька типів областей видимості:

- **глобальні змінні**: оголошені поза будь-якими функціями. Вони доступні у всій програмі.
- **локальні змінні**: оголошені всередині функцій або блоків коду. Вони доступні лише в межах цього блоку.

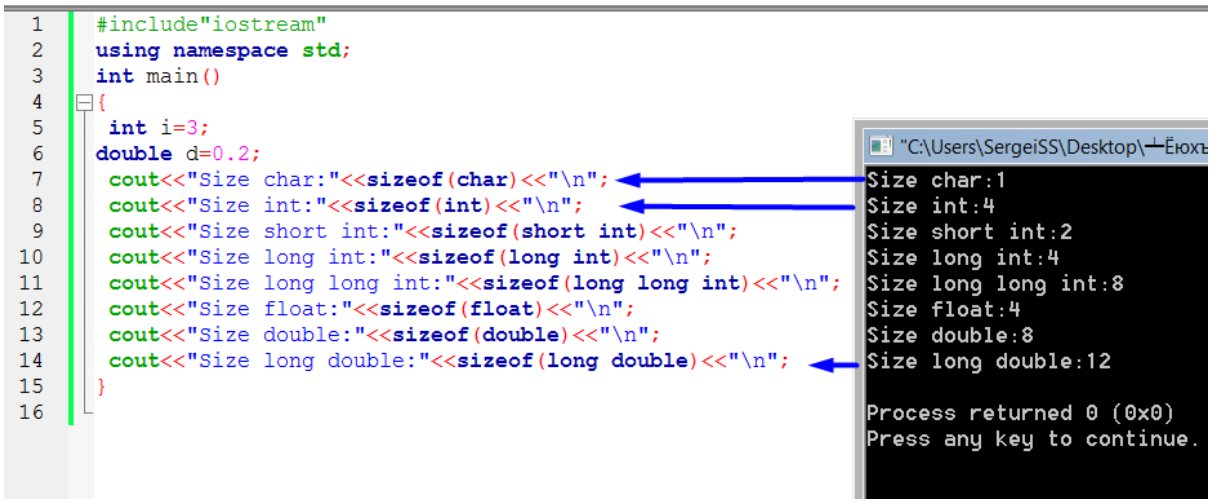
Наприклад:

```
1. int globalVar = 100;  // Глобальна змінна
2. int main() {
3.     int localVar = 50;  // Локальна змінна
4.     std::cout << globalVar << std::endl;  // Доступ до
глобальної змінної
5.     std::cout << localVar << std::endl;   // Доступ до
локальної змінної
6.     return 0;
7. }
```

Для зручності розуміння матеріалу можна розглянути приклад програми, завдання якої показати, якого розміру будуть зміні в байтах.

```
1. #include"iostream"
 2. using namespace std;
 3. int main()
 4. {
 5.  int i = 3;
 6.  double d = 0.2;
 7.  cout<<"Size char:"<<sizeof(char)<<"\n";
 8.  cout<<"Size int:"<<sizeof(int)<<"\n";
 9.  cout<<"Size short int:"<<sizeof(short int)<<"\n";
10.  cout<<"Size long int:"<<sizeof(long int)<<"\n";
11.  cout<<"Size long long int:"<<sizeof(long long int)<<"\n";
12.  cout<<"Size float:"<<sizeof(float)<<"\n";
13.  cout<<"Size double:"<<sizeof(double)<<"\n";
14.  cout<<"Size long double:"<<sizeof(long double)<<"\n";
15. }
```

Нижче наведено результат роботи програми, в якому видно, який розмір виділяється для кожного символу.



Арифметичні операції.

```
+ додавання
- віднімання
* множення
/ ділення, (10/3=3)
% остача від ділення, (13%4=1)
```

Операції порівняння.

```
== дорівнює
!= не дорівнює
< менше
> більше
<= менше або дорівнює
>= більше або дорівнює
```

Логічні операції.

```
&& логічне і
|| логічне або
! логічне НЕ
```

Операції виконання і присвоювання.

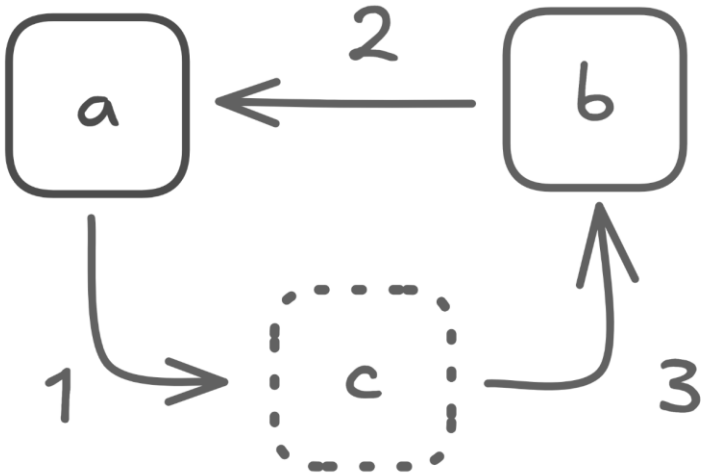
```
+=, -=, *=, /=, %=, |=, &=, ^=, <<=, >>=
```

Практичні завдання

Задача 1. Обмін значеннями

У двох комірках пам’яті проживають дві незалежні змінні **a** та **b**, кожна мріє, хоч на мить помінятись своїми значеннями зі своєю подругою та перевірити на зручність іншу комірку пам’яті. Допоможіть їм реалізувати задумане.

Ідея реалізації здається досить простою і може мати декілька різних рішень, одне з яких може бути задіяння додаткової змінної.



Псевдокод:

```
1. ПОЧАТИ
2. ВСТАНОВИТИ a = 10, b = 20
3. ВСТАНОВИТИ змінну c для збереження тимчасового значення
4. ЗБЕРЕГТИ значення a у змінну c
5. ПРИСВОЇТИ значення b змінній a
6. ПРИСВОЇТИ значення c змінній b
7. ВИВЕСТИ значення a і b
8. КІНЕЦЬ
```

C++ код:

```
 1. #include<iostream>
 2.
 3. using namespace std;
 4.
 5. int main()
 6. {
 7.  int a = 10, b = 20, c;
 8.  c = a; a = b; b = c;
 9.  cout << a << ' ' << b;
10. }
```

Задача 2. Зміна формату подання часу

Написати програму, яка перетворює час із хвилин у години та хвилини.

Технічні умови:

Зі стандартного потоку вводиться натуральне число N, яке визначає кількість хвилин. 0 < N < 10,000 (наприклад, 150).

Вихідні дані:

На стандартний вихід програма має вивести два числа: години та хвилини у форматі Хгод-Yхв. У вихідних даних неповинно бути пропусків, хвилини виводяться через тире.

Приклад:

Ввід	***Вивід***
150	2год-30хв

Пояснення ідеї розв'язку:

Ми знаємо, що 1 година = 60 хвилин. Отже, для перетворення кількості хвилин на години, потрібно поділити кількість хвилин на 60. Залишок від ділення на 60 дасть кількість хвилин, що залишилися.

Псевдокод:

```
1. ПОЧАТИ
2. ОТРИМАТИ значення n (кількість хвилин)
3. ОБЧИСЛИТИ кількість годин як n поділене на 60
4. ОБЧИСЛИТИ кількість хвилин як залишок від ділення n на 60
5. ВИВЕСТИ кількість годин та кількість хвилин
6. КІНЕЦЬ
```

C++ код:

```
1. #include <iostream>
2. #include <cstdlib>
3.
4. using namespace std;
5.
6. int main()
7. {
8.  int n;
9.  cin >> n;
```

```
10.  cout << n / 60 << "годин-" << n % 60 << "-хвилин";
11. }
```

Задача 3. Зміна запису числа

Допоможіть Петрику, який погано в школі вивчав математику написати програму, яка буде допомагати перетворювати дробові числа, записані десятковим числом у грошовий формат.

Технічні умови:

У стандартний потік вводиться одне число, записане у вигляді десяткового дробу, число з крапкою без розділових знаків 1<n< 100 000.

Вихідні дані:

Число у грошовому форматі з вказанням n грн. m коп. Округлене до сотої частини числа, вказане через пропуск.

Приклад:

Ввід	***Вивід***
12.5	12 грн 50 коп

Псевдокод:

```
1. ПОЧАТИ
2. ОТРИМАТИ число n (грошова сума з копійками)
3. ВИВЕСТИ цілу частину числа n як гривні
4. ОБЧИСЛИТИ залишок (копійки) як різницю між n і цілою частиною
n, помноживши на 100
5. ВИВЕСТИ залишок як копійки
6. КІНЕЦЬ
```

C++ код:

```
1. #include <iostream>
2. #include <cstdlib>
3. using namespace std;
4. int main()
5. {
6.  double n;
7.  cin >> n;
8.  cout << (int)n << " грн " << (n - (int)n) * 100 << "коп";
9. }
```

Задача 4. Сотні в 2-х чотиризначних

З клавіатури вводяться два додатних чотиризначних цілих числа, порахувати скільки у цих двох чисел разом є сотень.

Технічні умови:

З стандартного потоку вводяться два додатних чотиризначних числа.

Вихідні дані:

Виводиться одне, ціле число, це кількість сотень які є в першому і другому числі.

Приклад:

Ввід	***Вивід***
1234 1221	24
1000 1009	20

```
1. #include <iostream>
2. #include <cstdlib>
3. using namespace std;
4. int main()
5. {
6.   int a,b;
```

```
7.    cin >> a >> b;
8.    cout << a / 100 + b / 100;
9. }
```

Задача 5. Гаррі і числа через тире

Вивчаючи точні науки, в Хогвардсі юний Гаррі Поттер на першому курсі часто почав зустрічати числа дуже великого формату, і для нього було досить важко їх розуміти, саме тому професор запропонував цікаву схему, яка розбила вказане число у зручному для читання вигляді. У числа окремо виписані через тире — одиниці, десятки, сотні, тисячі і десятки тисяч.

Технічні умови:

З клавіатури задається число, відмінне від нуля і яке не перевищує 10 000.

Вихідні дані:

Цифри, які входять до складу введеного числа, записані через тире.

Приклад:

Ввід	***Вивід***
10000	1-0-0-0-0
23451	2-3-4-5-1

Псевдокод:

```
 1. ПОЧАТИ
 2. ОТРИМАТИ значення a
 3. ВИВЕСТИ результат ділення a на 10000, а потім додати "-"
 4. ОНОВИТИ a, взявши залишок від ділення на 10000
 5. ВИВЕСТИ результат ділення a на 1000, а потім додати "-"
 6. ОНОВИТИ a, взявши залишок від ділення на 1000
 7. ВИВЕСТИ результат ділення a на 100, а потім додати "-"
 8. ОНОВИТИ a, взявши залишок від ділення на 100
 9. ВИВЕСТИ результат ділення a на 10, а потім додати "-"
10. ОНОВИТИ a, взявши залишок від ділення на 10
11. ВИВЕСТИ a (останній залишок)
12. КІНЕЦЬ
```

C++ код:

```
 1. #include <iostream>
 2. #include <cstdlib>
 3.
 4. using namespace std;
 5. int main()
 6. {
 7.  int a;
 8.  cin >> a;
 9.  cout << a/10000 << "-";
10.  a %= 10000;
11.  cout << a/1000 << "-";
12.  a %= 1000;
13.  cout << a/100 << "-";
14.  a %= 100;
15.  cout << a/10 << "-";
16.  a %= 10;
17.  cout << a;
18. }
```

Задача 6. Шукаємо номер під'їзду

З’ясувати номер під’їзду та поверху по номеру квартири 9-ти поверхового будинку, вважаючи, що на кожному поверсі є 4 квартири, а нумерація квартир починається з першого під’їзду та відповідно першого поверху.

Технічні умови:

Зі стандартного потоку вводиться ціле число n: *0 < n <= 10 000.*

Вихідні дані:

У форматі «1п 1 поверх» записано під'їзд та поверх.

Приклад:

Ввід	***Вивід***
4	1п 1поверх
48	2п 3поверх

```
 1. #include <iostream>
 2. #include <cstdlib>
 3. using namespace std;
 4. int main()
 5. {
 6.  int a;
 7.  cin >> a;
 8.  cout << ((a - 1) / 36) + 1 << "п " << ((a - 1) % 36) / 4 +
1 << "поверх";
 9. }
```

Задача 7. Проєктуємо шахову дошку

Розмір шахової дошки задає кількість фігур на ній за простим правилом: кількість фігур, які розміщуються на полі, обов'язково має бути така, що повністю заповнює перший та другий ряд для кожного з супротивників, грає як правило тільки 2 гравців. Допоможіть майстрам розрахувати кількість фігур потрібних для гри на дошках довільної кількості клітинок.

Технічні умови:

Точно відомо, що дошки мають квадратну форму. Тому кількість клітинок завжди по вертикалі і горизонталі рівна. Найменша дошка може мати поле в 4 клітинки, на цьому полі може бути точно розміщено 4 фігури, для обох гравців.

Вихідні дані:

Майстер вказує одне число, кількість клітинок або по вертикалі, або по горизонталі 2<n<100.

Приклад:

Ввід	***Вивід***
2	4
5	20
8	32

```
 1. #include <iostream>
 2. #include <cstdlib>
 3.
 4. using namespace std;
 5. int main()
 6. {
 7.  int a;
 8.  cin >> a;
 9.  if (a < 4) cout << a * 2; else cout << a * 4; // задача
вирішується тільки умовою
10. }
```

Задача 8. Слова племені Макактус

Слова племені Макактус складаються з 5 літер кожне, в алфавіті використовуються 31 літера. Так сталося, що словами в їхній мові можна вважати всі можливі комбінації літер у цих словах.

Технічні умови:

Слова складаються тільки з 5 літер, кожна літера може бути однією з 31. Слово може починатися з великої або малої літери, але всі наступні – лише малі.

Вихідні дані:

Одне число, яке є загальною кількістю слів усієї мови макактус.

```
1. #include <iostream>
2. #include <cstdlib>
3.
4. using namespace std;
5. int main()
6. {
7.  cout << 62 * 31 * 31 * 31 * 31;
8. }
```

Задача 9. Флешка і файли

Є флешка обсягом 4 гібібайти. На флешку записується інформація, це 4 текстові файли, розмір кожного файлу записується з клавіатури у вигляді мегабайт і не перевищує 1 гібібайт кожен, з'ясувати скільки вільного місця залишилося на флешці, після запису цих 4-х файлів.

Примітка: за стандартом ДСТУ IEC 80000-13:2016 (міжнародний IEC 80000-13:2008, IDT), 1 гігабайт містить 1000 мегабайт, а 1 гібібайт = 1024 мебібайти.

Технічні умови:

З стандартного потоку вводиться розміри 4 файлів, вважаємо, що всі розміри вказуються в мебібайтах.

Вихідні дані:

Одне число, кількість мебібайт, які залишаються вільними для запису. Результат округлити до десятитисячних.

```cpp
#include <iostream>
#include <cstdlib>

using namespace std;
int main()
{
 int q, w, e, r;
 cin >> q >> w >> e >> r;
 cout << 4096 - q - w - e - r;
}
```

3. Розгалуження та логіка

Досить часто доводиться використовувати складений оператор, який групує кілька окремих операторів (інструкції) та дозволяє їх використовувати як єдине ціле. Для об'єднання таких інструкцій використовуються фігурні дужки { }.

```
{
 //блок інструкцій
   оператор_1;
   оператор_2;
   .
   .
   .
   оператор_n;
 }
```

Використовуються складені оператори в конструкціях if та else.

Якщо доводиться використати більше одного оператора, то використовується складений оператор.

Вказівка розгалудження If дозволяє обирати один із варіантів дій.

```
//загальний вигляд повної форми
if (умова) оператор1 else оператор2
//загальний вигляд короткої форми
if (умова) оператор1;

//приклад
//if (x > y) a = x; else a = y;
//змінній a присвоюється більше з двох чисел

// якщо потрібно виконати декілька інструкції програми використовують групування операторів фігурними дужками
if (x < 0){
    x = -x;
    cout << "Якщо треба зробити від'ємну змінну x додатною";
 }

//Для перевірки декількох умов можна використати декілька else if
if (x < 0)
```

```
18.     cout << "Від’ємне значення";
19. else if (x > 0)
20.     cout << "Додатне значення";
21. else
22.     cout << "Нуль";
```

Оператор вибору **switch** використовується у випадку, коли для кожного зі значень виразу потрібно виконати певні дії.

```
 1. switch ( <зміна> ) {
 2. case значення 1:
 3.   Виконати якщо <змінна> == значення 1
 4.   break;
 5. case значеня 2:
 6.   Виконати якщо <змінна> == значення 2
 7.   break;
 8. ...
 9. default:
10.   виконується у випадку коли жоден варіант не підходить
11.   break;
12. }
13. //значення в case можуть бути лише константами!
```

У попередньому прикладі з’являється новий оператор **break**.

Оператор **break** використовується для дострокового завершення виконання циклів та конструкцій **switch**. Він дозволяє вийти з циклу або блоку коду, навіть якщо умова, що визначає його завершення, ще не виконана.

Оператор break зазвичай застосовується в циклах (for, while, do-while) для дострокового виходу з циклу, коли досягнуто певної умови.

Наприклад:

```
1. #include <iostream>
2.
3. int main() {
4.     for (int i = 0; i < 10; i++) {
5.         if (i == 5) {
6.             break;
7.         }
```

```
 8.         std::cout << "i: " << i << std::endl;
 9.     }
10.     std::cout << "Цикл завершено" << std::endl;
11.     return 0;
12. }
```

Наступні дві програми виконують одну й ту ж роботу, але при цьому задіяні різні конструкції і**f** та **switch**.

```
1. if (code == 0) { cout << "це нуль"; x = 0;}
2. else if (code == 1) { cout << "це один"; x = 1;}
3. else if (code == 2) { cout << "це два"; x = 2;}
4. else {   cout << " Неправильне значення "; }
```

```
1. switch (code) {
2. case 0: cout << "це нуль"; x = 0; break;
3. case 1: cout << "це один"; x = 1; break;
4. case 2: cout << "це два";  x = 2; break;
5. default:
6.     cout << "Неправильне значення";
7. }
```

Практичні завдання

Задача 1. Більше з двох натуральних чисел

Посперечалися якось два натуральних числа, яке з них більше. Допоможіть цим числам, написавши програму, що виводила б більше з них на екран.

Якщо числа рівні, виводити повідомлення: “вони рівні”.

Технічні умови:

Зі стандартного потоку вводиться 2 натуральних числа N та M.

Вихідні дані:

На стандартний вихід програма має вивести рядок “вони рівні”, якщо числа однакові або найбільше з двох чисел.

Приклад:

Ввід	***Вивід***
2 5	5
10 10	вони рівні

Псевдокод:

```
ПОЧАТИ
   ОТРИМАТИ значення a і b
   ЯКЩО a дорівнює b
      ВИВЕСТИ "вони рівні"
   ІНАКШЕ
      ЯКЩО a більше ніж b
         ВИВЕСТИ a
      ІНАКШЕ
         ВИВЕСТИ b
КІНЕЦЬ
```

C++ код:

```
#include <iostream>

using namespace std;

int main() {
  int a, b;
  cin >> a >> b;

  if (a == b) {
    cout << "вони рівні";
  }
else
{
    if (a > b) {
      cout << a;
    }
 else
```

```
18. {
19.         cout << b;
20.      }
21.    }
22. }
```

Задача 2. Більше з трьох натуральних чисел

Чисел з попередньої задачі вже стало три, вони всі натуральні і почали суперечку, хто з них більший.

Технічні умови:

Зі стандартного вхідного потоку вводяться три цілих числа, кожне в окремому рядку.

Вихідні дані:

У стандартний вихідний потік вивести більше з чисел. Якщо вони однакові - вивести повідомлення: “всі рівні”.

Псевдокод:

```
 1. ПОЧАТИ
 2.    ОТРИМАТИ значення a, b, c
 3.    ЯКЩО a дорівнює b і b дорівнює c
 4.       ВИВЕСТИ "вони рівні"
 5.    ІНАКШЕ
 6.       ЯКЩО a більше або рівне b і більше або рівне c
 7.          ВИВЕСТИ a
 8.       ІНАКШЕ
 9.          ЯКЩО b більше або рівне a і більше або рівне c
10.             ВИВЕСТИ b
11.          ІНАКШЕ
12.             ЯКЩО c більше/рівне a і більше/рівне b
13.                ВИВЕСТИ c
14. КІНЕЦЬ
```

C++ код:

```cpp
#include <iostream>
#include <cmath>

using namespace std;

int main() {
  int a, b, c;
  cin >> a >> b >> c;
  if (a == b && b == c) cout << "вони рівні";
  else
  if (a >= b && a >= c) cout << a;
  else
  if (b >= a && b >= c) cout << b;
  else
  if (c >= b && c >= a) cout << c;
}
```

C++ код (оптимізований):

```cpp
#include <iostream>

using namespace std;

int main() {
  long long a, b, c, m;
  cin >> a >> b >> c;

  if (a == b && b == c) {
    cout << "всі рівні";
  } else {
    m = a;
    if (b > m) m = b;
    if (c > m) m = c;

    cout << m;
  }
}
```

Задача 3. Знайти двох братів-однолітків

По лісі бродить хитрий заєць з планшетом, шукає трьох поросят, щоб перевірити, чи є серед них хоча б двоє з однаковою повною кількістю років.

Довговухий абсолютно не знає, як порівнювати числа, тому використовує для цього програму, яку написав вовк, яка за даними введеними до програми перевіряє, чи є брати поросята, які мають однакову кількість років.

Технічні умови:

Зі стандартного вхідного потоку вводяться три цілих числа, кожне в окремому рядку.

Вихідні дані:

У стандартний вихідний потік вивести TRUE, якщо рівні тільки два з них і FALSE в іншому випадку. Якщо всі брати поросята мають однаковий вік, то також виводити виводити TRUE.

```
 1. #include <iostream>
 2.
 3. using namespace std;
 4.
 5. int main() {
 6.   long long a, b, c;
 7.   cin >> a >> b >> c;
 8.
 9.   if (a == b || b == c || a == c) {
10.     cout << "TRUE";
11.   } else {
12.     cout << "FALSE";
13.   }
14. }
```

Задача 4. Пошук щасливих цифр у записі числа

Щасливим є те число: до складу якого входить цифра 7. Перевірити, чи входить до складу цифр будь-якого п’ятизначного числа, введеного з клавіатури, хоч одна сімка.

Технічні умови:

Зі стандартного вхідного потоку вводиться ціле п’ятизначне число.

Вихідні дані:

У стандартний вихідний потік вивести TRUE, якщо десятковий запис містить цифру 7 і FALSE, якщо ні.

```
 1. #include <iostream>
 2.
 3. using namespace std;
 4.
 5. int main() {
 6.   int x;
 7.   bool flag = false;
 8.   cin >> x;
 9.
10.   for (int i = 0; i < 5 && !flag; i++) {
11.     flag = x % 10 == 7;
12.     x /= 10;
13.   }
14.
15.   if (flag) {
16.     cout << "TRUE";
17.   } else {
18.     cout << "FALSE";
19.   }
20. }
```

Так-так, ми бачимо, що тут є незнайома тобі конструкція for(...). Можемо по секрету сказати, що це цикл, який буде розглядатися в наступному розділі. А зараз, поглянемо на рішення цієї задачі без циклу.

```
1. #include <iostream>
2.
```

```
 3. using namespace std;
 4.
 5. int main() {
 6.   int x;
 7.   cin >> x;
 8.
 9.   int a, b, c, d, e
10.   a = x / 10000;
11.   b = (x / 1000) % 10;
12.   c = (x / 100) % 10;
13.   d = (x / 10) % 10;
14.   e = x % 10;
15.
16.   if (a == 7 || b == 7 || c == 7 || d == 7 || e == 7) {
17.     cout << "TRUE";
18.   } else {
19.     cout << "FALSE";
20.   }
21. }
```

Виглядає не надто “програмістично”, правда ж? При збільшенні кількості цифр числа треба буде збільшувати кількість змінних та дописувати величезну перевірку, а якщо дозволити користувачу вводити довільне число, то задача взагалі стає нерозв’язною без циклів, у них вся сила!

Задача 5. Вовк і трикутники

Вовк захотів у своєму лігві за зиму прикрасити стіну різними трикутниками.

Всі зайці в лісі по черзі гризуть тоненькі деревцята, щоб принести кожен по три жердини різної довжини, з яких має будуватись трикутник.

Зайці не здійснюють ніяких вимірів, тому гризуть всі жердинки, якої попало довжини, а миші біля берлоги обгризають кінці таким чином, що вовк отримує такі палиці, які мають цілі довжини сторін певної метричної системи.

Допоможіть вовку написати програму, яка перевірятиме за довжинами сторін введеними з клавіатури в стовпець, чи можна скласти трикутник із введеними довжинами.

Розумна сова підказала, що варто здійснювати ще додатково перевірку, чи є трикутник рівносторонній чи різносторонній у випадку, якщо він існує.

Вовк користується наступними правилами:

- якщо трикутник рівносторонній - виводиться 1;
- якщо трикутник має різні сторони - виводиться 2;
- якщо трикутника з такими сторонами немає - виводиться 0.

Технічні умови:

Зі стандартного вхідного потоку вводятся три числа, кожне в окремому рядку.

Вихідні дані:

У стандартний вихідний потік вивести одне з чисел: 0, 1, 2.

Приклад:

Ввід	***Вивід***
3 3 3	1
1 1 10	0
3 4 5	2

```
1. #include <iostream>
2. #include <cmath>
3.
```

```
 4. using namespace std;
 5.
 6. int main() {
 7.   int a, b, c;
 8.   cin >> a >> b >> c;
 9.
10.   if (a + b <= c || b + c <= a || a + c <= b) {
11.    cout << 0;
12.   } else if (a == b && a == c) {
13.    cout << 3;
14.   } else if (a == b || a == c || b == c) {
15.     cout << 2;
16.   } else {
17.     cout << 1;
18.   }
19. }
```

4. Циклічні алгоритми

Цикли дозволяють виконувати певний блок коду кілька разів до тих пір, поки виконується певна умова. Це важливий елемент керування потоком програми, який дозволяє автоматизувати повторювані дії.

C++ використовує три структури циклів:

- **for** — використовується, коли кількість ітерацій відома заздалегідь;

```
1. for (лічильник = значення; лічильник < значення; крок циклу)
2. {
3.     // тіло циклу
4. }
```

- **while** — використовується, коли потрібно повторювати блок коду до тих пір, поки виконується умова;

```
1. while(умова)
2. {
3.     // тіло циклу
4. }
```

- **do-while** — схожий на while, але блок коду виконати хоча б один раз, незалежно від умови.

```
1. do
2. {
3.     // тіло циклу
4. }
5. while(умова);
```

Для припинення роботи циклу застосовують команди break (команда відразу припиняє роботу циклу, не даючи завершитись ітерації).

Наприклад:

```
1. if (i > 9) break;
```

Ітерація – процес повторного виконання блоку коду кілька разів з різними значеннями або умовами.

У випадку, коли потрібно завершити ітерацію циклу, використовують команду **continue**, що завершує всі команди, вкладені до циклу, та здійснює перехід на наступну ітерацію.

Практичні завдання

Задача 1. Таблиця множення

Вивести на екран таблицю множення на N чисел від 1 до M рядків.

Технічні умови:

У стандартний потік вводяться два натуральні числа: *0 < N, M < 100*

Вихідні дані:

На екран виводяться ряди таблиці множення.

```
#include <bits/stdc++.h>
#include <iostream>
#include <cstdlib>

using namespace std;

int main()
{
    int n,m;
    cin >> n >> m;

    for(int i = 1; i < m; i++)
    {
        cout << i << " * " << n << " = " << i * n << endl;
    }
}
```

Задача 2. Квадрати та куби

Написати програму, яка виводить в рядок квадрати та куби N чисел на екран.

Технічні умови:

У стандартний потік вводиться натуральне число, у рядок вивести всі квадрати і куби чисел до заданого числа N.

Вихідні дані:

На екран виводяться ряди чисел, першим у ряді являється індекс порядкового номера включно до заданого з клавіатури числа. Другим в кожному ряді є квадрат індексу і наступне число куб для кожного з чисел.

Приклад:

Ввід	***Вивід***
3	1 1 1 2 4 8 3 9 27

```
 1. #include <iostream>
 2. #include <cstdlib>
 3.
 4. using namespace std;
 5. int main()
 6. {
 7.     int i, n;
 8.     cin >> n;
 9.     for (i = 1; i <= n; i++)
10.         cout << i << " " << i * i << " " << i * i * i <<
endl;
11. }
```

Задача 3. Сума перших натуральних

Допоможіть учню Петрику порахувати суму перших N цілих додатних парних чисел.

Технічні умови:

Вивести на екран суму чисел, які задовольняють дану умову: *0 < N < 10 000.*

Вихідні дані:

Одне число, яке є сумою всіх парних цілих чисел, у заданій послідовності.

Приклад:

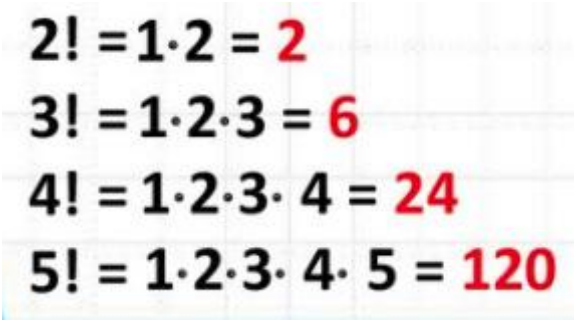
Ввід	***Вивід***
5	30

```
#include <iostream>
#include <cstdlib>

using namespace std;
int main()
{
    int n, c = 1, s = 0, i;
    i = 1;
    cin >> n;
    while (c <= n){
        if (i % 2 == 0) {
            s += i;
            c++;
        }
        i++;
    }
    cout << s;
}
```

Задача 4. Цей унікальний факторіал

Порахуйте факторіал числа, введеного з клавіатури.



Технічні умови:

Факторіалом числа n називають добуток цілих чисел від 0 до n. 0<n<100.

Вихідні дані:

Одне число, яке є факторіалом заданого з клавіатури.

Приклад:

Ввід	***Вивід***
5	120

```
 1. #include <iostream>
 2. #include <cstdlib>
 3.
 4. using namespace std;
 5.
 6. int main()
 7. {
 8.     int n, i, f = 1;
 9.     cin >> n;
10.     for (i = 2; i <= n; i++) f*=i;
11.     cout << f;
12. }
```

Задача 5. Таблиця Піфагора

Таблиця Піфагора — це таблиця, де кожен елемент є результатом множення чисел у відповідному рядку та стовпці. Виведіть на екран таблицю Піфагора.

Технічні умови:

Програма повинна відтворювати роботу таблиці множення (таблиця Піфагора) для чисел від 1 до 9.

Вихідні дані:

На екран виводиться таблиця, яка виглядає як таблиця Піфагора з числовими результатами множення від 1 до 9.

	1	2	3	4	5	6	7	8	9
1	1	2	3	4	5	6	7	8	9
2	2	4	6	8	10	12	14	16	18
3	3	6	9	12	15	18	21	24	27
4	4	8	12	16	20	24	28	32	36
5	5	10	15	20	25	30	35	40	45
6	6	12	18	24	30	36	42	48	54
7	7	14	21	28	35	42	49	56	63
8	8	16	24	32	40	48	56	64	72
9	9	18	27	36	45	54	63	72	81

Псевдокод:

```
1. ПОЧАТИ
2.    ДЛЯ КОЖНОГО i від 1 до 9
3.       ДЛЯ КОЖНОГО j від 1 до 9
4.          ВИВЕСТИ результат множення j на i, додаючи табуляцію
5.       ПЕРЕЙТИ на новий рядок
6. КІНЕЦЬ
```

C++ код:

```
1. #include <iostream>
2. #include <cstdlib>
3.
4. using namespace std;
5. int main()
6. {
7.     for (int i = 1; i <= 9; i++) {
```

```
 8.         for (int j = 1; j <= 9; j++)
 9.             cout << j * i << "\t";
10.         cout << endl;
11.     }
12. }
```

Задача 6. Піноккіо і кількість цифр

Багатий Піноккіо вступив у школу програмістів, і одним з екзаменаційних питань для вступу було з'ясовувати: скільки цифр має введене з клавіатури число.

Технічні умови:

Число не перевищує один мільйон.

Вихідні дані:

Число, яке вказує з скількох цифр складається число.

Приклад:

Ввід	***Вивід***
234567	6

```
 1. #include <iostream>
 2. #include <cstdlib>
 3.
 4. using namespace std;
 5. int main()
 6. {
 7.     int i, c = 0;
 8.     cin >> i;
 9.     while (i != 0) {
10.         c++;
11.         i /= 10;
12.     }
13.     cout << c;
14. }
```

Задача 7. Пароль і шпигуни

Шпигуни сиділи над розробкою нового секретного проєкту. Їм на думку спала геніальна ідея поставити пароль на вхід до системи. Але знання програмування в них було не на найвищому рівні, тому вони попросили допомоги.

Система представляє собою виведення в чотирьох стовпцях випадкових чисел, рівно по 5 рядків кожний. А для перегляду чисел потрібно ввести правильний пароль.

Технічні умови:

Зі стандартного потоку вводиться пароль, це просте 4 значне число, якщо пароль є вірним (наприклад, 1313), то відбувається вхід у систему, яка виводить на екран випадкові числа, а якщо вводиться інше число, яке відмінне від паролю, то виводиться повідомлення «error! key password->» та просить повторне введення паролю.

Якщо пароль вводиться більше ніж 10 разів, система блокується і програма завершує свою роботу з відповідним повідомленням: «gudbai!!!»

Вихідні дані:

Вихідними даними є можливість побачити ряди випадкових чисел, при введенні неправильного паролю.

Приклад:

Ввід	***Вивід***
1313	виведення випадкових чисел
1234	error! key password->

```
1. #include <iostream>
2. #include <cstdlib>
3.
4. using namespace std;
5. int main() // password - 1313
```

```
 6. {
 7.     const int pass=1313;
 8.     int n,c=0;
 9.     while (cin >> n) {
10.         c++;
11.         if (n==pass) {
12.             for (int i=1;i<=5;i++) {
13.                 for (int j=1;j<=4;j++) cout << rand() << "
";
14.                 cout << endl;
15.             }
16.             break;
17.         }
18.         else {
19.
20.             if(c<10) cout << "error! key password->"; else{
21.                 cout << "gudbai!!!";
22.                 break;
23.             }
24.
25.         }
26.     }
27. }
```

Ну що, читачу, як тобі такий код? Знаємо-знаємо, його опанувати дуже важко, а що найстрашніше, то схожі помилки можна часто побачити десь на просторах інтернету. Тому закликаємо дотримуватися хорошого стилю коду: використання пробілів та відступів, блоків коду та конкретних назв змінних. Поглянь на наступний, він виконує ті ж самі функції, проте є більш читабельним та правильним з точки зору хорошого тону програмування.

```
1. #include <iostream>
2. #include <cstdlib>
3.
4. using namespace std;
5.
6. int main() {
7.   const int pass = 1313; // password - 1313
8.
9.   int n;
```

```
10.    int count = 0;
11.
12.    bool go = true;
13.
14.    while (go && (cin >> n)) {
15.      count++;
16.
17.      if (n == pass) {
18.        for (int i = 1; i <= 5; i++) {
19.          for (int j = 1; j <= 4; j++) {
20.            cout << rand() << " ";
21.          }
22.          cout << endl;
23.        }
24.
25.        go = false;
26.      } else {
27.        if (count < 10) {
28.          cout << "error! key password -> ";
29.        } else {
30.          cout << "gudbai!!!";
31.          go = false;
32.        }
33.      }
34.    }
35. }
```

Задача 8. Числа чарівниці одиниці

Є у світі математики числа, справжні чарівники, вони іноді здатні змінювати свою структуру і сутність. Ваша задача написати алгоритм, який задане з клавіатури 4-х значне число буде заміняти на геть інше число, яке складається тільки з одиниць.

Кожна цифра числа, перетворюється у відповідну кількість одиниць, і готове число виводиться на екран.

Технічні умови:

З клавіатури вводиться будь-яке 4-х значне число. Кожна цифра цього числа відповідно замінюється на кількість одиничок.

Вихідні дані:

Результуюче число, яке складається тільки з одиничок.

Приклад:

Ввід	***Вивід***
1231	1111111
4523	11111111111111
1000	1

Псевдокод:

```
1. ПОЧАТИ
2.    ОТРИМАТИ значення n
3.    ПОКИ n не дорівнює 0
4.       ОТРИМАТИ останню цифру c як залишок від ділення n на 10
5.       ДЛЯ кожного i від 0 до c-1
6.          ВИВЕСТИ 1
7.       ОНОВИТИ n, поділивши його на 10
8. КІНЕЦЬ
```

C++ код:

```
 1. #include <iostream>
 2. #include <cstdlib>
 3.
 4. using namespace std;
 5. int main()
 6. {
 7.     int n,c;
 8.     cin >> n;
 9.     while (n != 0) {
10.         c = n % 10;
11.         for (int i = 0; i < c; i++) cout << 1;
12.         n /= 10;
13.     }
14. }
```

Задача 9. Числа липучки

Число липучка — це новий термін в теорії чисел, які вивчаються в Хогвардсі. Зміст оригінальності цього числа полягає в тому що воно постійно змінює свій вигляд.

З самого початку це число завжди являє собою одиницю, але вже після декількох трансформацій стає принципово іншим числом, яке і за розміром і за своєю структурою нагадує липучку, тобто число, яке притягує до себе всі інші числа.

Технічні умови:

У системі зберігається число 1, воно завжди задає перше число результату.

Із стандартного потоку вводиться 4 будь-які числа, кількість цифр в яких не перевищує 3, якщо цифр більше за 3, то число просить система ввести повторно. Кожне число записується до попереднього за правилом «прилипаю за попереднім».

Вихідні дані:

Виводиться на екран число «липучка», яке набрало до себе всі введені з клавіатури 3-цифрові числа.

Приклад:

Ввід	***Вивід***
23 35 123 9	123351239
12 12 2334 12 12	112121212

```
1. #include <iostream>
2. #include <cstdlib>
```

```
 3. using namespace std;
 4. int main()
 5. {
 6.     int n, a, b, c, d;
 7.     cin >> a;
 8.     while (a / 1000 != 0) cin >> a;
 9.     cin >> b;
10.     while (b / 1000 != 0) cin >> b;
11.     cin >> c;
12.     while (c / 1000 != 0) cin >> c;
13.     cin >> d;
14.     while (d / 1000 != 0) cin >> d;
15.     cout << 1 << a << b << c << d;
16. }
```

5. Функції

Функція користувача – поіменована група команд, оголошена у файлі заголовків (або в основній програмі). Функції дозволяють розбивати програму на менші, незалежні частини з чітко визначеним призначенням, що спрощує її розуміння, тестування та повторне використання коду.

Використання функцій у програмах дозволяє реалізувати цілу низку переваг:

- модульність - задача об'єднується в логічні блоки, і це надає можливість легше її налагоджувати та шукати помилки;
- можливість повторного використання - написана частина коду у вигляді функцій може використовуватись потрібну кількість разів без дублювання коду;
- інкапсуляція - можливість приховувати деталі розробки працюючи з інтерфейсом програми, що є одним з основних принципів об'єктно-орієнтованого програмування.

Функція додавання трьох цілих чисел:

```
1. int sum(int a, int b, int c)
2. {
3.  int result;
4.  result = a + b + c;
5.  return result;
6. }
```

Функція цілком самостійний блок програми. Під час виклику функції їй передаються певні аргументи, та після виконання описаних у ній дій повертається результат.

Будь-яка програма написана мовою C++ вже містить принаймні одну функцію main(). Саме з неї починається виконання основної програми.

Принцип роботи функції полягає в тому що, як тільки керування програмою передається у функцію, починає відпрацьовувати її тіло (виконуються всі команди функції), після цього програма повертає в точку входу те значення, яке було отримано в ході роботи функції.

Практичні завдання

Задача 1. Веселий геодезист

Допоможіть геодезисту Петрику написати функцію для його розумного робота-помічника, який розуміє всі команди лише в синтаксисі C++. Завдання функції - обчислювати площу трикутника за заданими сторонами st1, st2, st3.

Робот має вміти перевіряти чи можна побудувати такий трикутник, який вкаже користувач довжинами його відповідних сторін, які, у свою чергу, задаються координатами вершин відповідно.

Технічні умови:

Зі стандартного потоку вводиться 6 чисел - X та У координати трьох вершин.

Вихідні дані:

На стандартний вихід програма має вивести одне число - площу трикутника або повідомлення про помилку.

Приклад:

Ввід	***Вивід***
0 0 0 3 4 0	6
0 0 0 0 0 1	Error! 0

```cpp
#include <iostream>
#include <cmath> // бібліотека для роботи з математичними функціями

using namespace std;

double length(int x1, int y1, int x2, int y2) { // функція для знаходження відстані між двома точками
  return sqrt(pow(x1 - x2, 2) + pow(y1 - y2, 2));
}

double area(double a, double b, double c) { // функція для знаходження площа трикутника за 3 сторонами
  if (a >= b + c || b >= a + c || c >= a + b) {
    cout << "Error!" << endl;
    return 0;
  }

  double p, s;
  p = (a + b + c) / 2.0;
  s = sqrt(p * (p - a) * (p - b) * (p - c));
  return s;
}

int main() {
  int x1, y1, x2, y2, x3, y3; // набір координат точок

  cin >> x1 >> y1 >> x2 >> y2 >> x3 >> y3;

  // обчислюємо відстані між точками
  int a = length(x1, y1, x2, y2);
  int b = length(x1, y1, x3, y3);
  int c = length(x3, y3, x2, y2);

  // обчислюємо та виводимо площу
  cout << area(a, b, c);
}
```

Задача 2. Функція формування паролю

Сергій з Максимом вирішили розробити надійну систему захисту паролей шкільних акаунтів пошти та реалізувати ці завдання, написавши функцію користувача, щоб інші програмісти могли використати їхній алгоритм.

Основні вимоги це те, щоб пароль генерувався за наступними правилами:

- пароль завжди містив 7 цифр;
- формування чисел починається з введення користувачем його улюбленого семизначного числа;
- після цього до цього числа додається його точна копія;
- всі числа, які отримуються після утворення 8 числа (хвоста з права) відсікаються без округлення;
- користувачем з консолі вводиться ще одне число, день його народження;
- це число вказує скільки разів потрібно запустити ще додавання й обрізання отриманого результату.

```
 1. #include <iostream>
 2. #include <cstdlib>
 3.
 4. using namespace std;
 5.
 6. int password(int number, int day) {
 7.   long result = number;
 8.
 9.   for (int i = 0; i < day; i++) {
10.     result += number;
11.     if (result >= 10000000) {
12.       result /= 10;
13.     }
14.   }
15.
16.   return result;
17. }
18.
19. int main() {
20.   int number, day;
21.   cin >> number >> day;
22.
23.   cout << password(number, day);
24. }
```

6. Масиви

Масив - пойменована скінченна послідовність величин одного типу, доступ до яких можна отримати за їх порядковим номером (індексом).

```
 1.  //тип ім’я_масиву [розмір];
 2. розмірність - це кількість елементів в масиві.
 3.
 4. //опис масиву x із 10 цілих чисел
 5. int x[10];
 6.
 7. //опис масиву із 20 дійсних чисел.
 8. float a [10];
 9.
10. //описана  константа та масив з 20 дійсних чисел
11. const int n=20;
12. double B[n];
13. Sum=B[0]+B[n-1] ; //сума першого та останнього елементів
масиву.
14.
15.
16.  //Елементи масиву в С++ нумеруються з нуля.
17.  //Перший елемент завжди має індекс нуль.
18. // Індекс останнього елементу на одиницю менше заданої при
його опису розмірності.
19.
20. // масив символів, елементи якого нумеруються від 0 до 4.
21. Char C[5];
22.
23. float a[6]= {1.2,(float)3/4,5./6,6.1}; /*формується масив із
шести дійсних чисел, значення елементам присвоюється по
порядку,елементи,значення яких не вказані, обнуляються. */
```

Практичні завдання

Задача 1. Суми в масиві для Колобка

(Завдання на обчислення суми елементів масиву, пошук найбільшого та найменшого значення)

Колобок планує створити масив із N (N < 200) різних цілих чисел для того, щоб зберігати там інформацію про свої заробітки у діда. Іноді він заробляє досить непогано, а іноді навіть може бути винен йому гроші. Допоможіть порахувати суму, яка буде зберігатись у його масиві, вивести найменшу і найбільшу зароблену зарплату.

Технічні умови:

Перший рядок вхідного стандартного потоку містить натуральне число n (n<200). У другому рядку записані через пропуск n цілих чисел, які по модулю не перевищують 1000.

Вихідні дані:

У стандартний потік записати всю зароблену суму, а також вивести найменшу і найбільшу разову зарплату, яка зберігалась в масиві.

Приклад:

Ввід	***Вивід***
10 1 21 -5 6 4 7 18 9 5 10	76 -5 21

```
1. #include <iostream>
2.
3. using namespace std;
4.
5. int main()
6. {
```

```
 7.   int a[200], b[200];
 8. #include <iostream>
 9.
10. using namespace std;
11.
12. int main()
13. {
14.   int a[200];
15.   int n, sum = 0;
16.
17.   cin >> n;
18.   for (int i = 0; i < n; i++) {
19.     cin >> a[i];
20.     sum += a[i];
21.   }
22.
23.   int nma=a[0], nmb=a[0];
24.
25.   for(int i = 0; i < n; i++){
26.     if(a[i] < nma){
27.       nma = a[i];
28.     } else if(a[i] > nmb){
29.       nmb = a[i];
30.     }
31.   }
32.
33.   cout << sum << " " << nma << " " << nmb << endl;
34. }
```

Задача 2. Катрусині масиви і унікальні мінімальні

(Завдання на пошук заданого числа в декількох масивах за ключовими умовами)

Катруся має два масиви по k елементів у кожному. Вона планує знайти найменший елемент у першому масиві та перевірити, чи міститься такий елемент у другому масиві.

Технічні умови:

Перший рядок містить натуральне число N (N <= 100). У другий і третій рядки містять N натуральних чисел, які не більше 1000.

Вихідні дані:

Якщо таке число існує, виведіть його. Інакше вивести "NO" (без лапок).

Приклад:

Ввід	***Вивід***
3 1 2 3 3 4 5	NO
3 1 2 3 1 4 5	1

```
 1. #include <iostream>
 2. using namespace std;
 3.
 4. int main() {
 5.     int n;
 6.     cin >> n;
 7.
 8.     int a[100], b[100];  // Масиви для зберігання до 100
елементів
 9.
10.     // Введення першого масиву
11.     for (int i = 0; i < n; i++) {
12.         cin >> a[i];
13.     }
14.
15.     // Введення другого масиву
16.     for (int i = 0; i < n; i++) {
17.         cin >> b[i];
18.     }
19.
20.     // Пошук найменшого елемента в першому масиві
21.     int min = a[0];
```

```
22.     for (int i = 1; i < n; i++) {  // Починаємо з 1,
оскільки 0-й вже є мінімумом
23.         if (a[i] < min) {
24.             min = a[i];
25.         }
26.     }
27.
28.     // Перевірка, чи міститься найменший елемент у другому
масиві
29.     bool result = false;
30.     for (int i = 0; i < n; i++) {
31.         if (b[i] == min) {
32.             result = true;
33.             break;
34.         }
35.     }
36.
37.     // Виведення результату
38.     if (result) {
39.         cout << min;
40.     } else {
41.         cout << "NO";
42.     }
43.
44.     return 0;
45. }
```

Задача 3. Монети в Хогвардсі

(Завдання на злиття одновимірних масивів)

Герміона і Гаррі мають N та M (0 < N, M < 10) різних комірок у сховищі Хогвартса для зберігання золотих монет, кожна комірка може зберігати цілу кількість монет, яка не перевищує 1000. Рон має доступ до всіх комірок Герміони та Гаррі і планує перерахувати, скільки він отримає всього монет з обох сховищ. А ще він планує найняти собі у власність M+N комірок і скласти так монети, щоб вони були розміщені в строгому порядку по кількості, починаючи від комірки, в якій лежатиме найбільша кількість і аж до найменшої.

Вихідні дані:

Вихідні дані містять загальну суму в першому рядку та відсортований за спаданням масив.

Приклад:

Ввід	***Вивід***
3 3 3 2 1 2 4 5	17 5 4 3 2 2 1

```
 1. using namespace std;
 2.
 3. int main()
 4. {
 5.   int c[20];
 6.   int n, m, r;
 7.   int sum = 0;
 8.
 9.   cin >> n >> m;
10.
11.   r = n + m;
12.   for (int i = 0; i < r; i++) {
13.     cin >> c[i];
14.     sum += c[i];
15.   }
16.
17.   for (int i = 0; i < r - 1; i++) {
18.     for (int j = 0; j < r - i - 1; j++) {
19.       if (c[j] < c[j + 1]) {
20.         int temp = c[j];
21.         c[j] = c[j + 1];
22.         c[j + 1] = temp;
23.       }
24.     }
25.   }
26.
27.   cout << sum << endl;
28.   for(int i = 0; i < r; i++){
29.     cout << c[i] << " ";
30.   }
```

```
31.
32.    return 0;
33. }
```

Задача 4. Пошук популярних чисел

Калькулятор, створений учнем Сергійком, дозволяє вводити цілі числа, розділяючи їх пропусками. Виконуючи обчислення лабораторної роботи, хлопцеві доводиться вводити досить немало різних чисел. Втомившись від розрахунків, Сергійко замислився, яку цифру йому довелось вводити найбільшу кількість разів.

У випадку, коли шукані числа повторюються, виводити всі їх у порядку зростання.

Технічні умови:

У першому рядку вхідного потоку записано кількість чисел N ($0 < N \leq 10000$), у наступному рядку записано самі числа через пропуск, які не перевищують за модулем 10000.

Вихідні дані:

У вихідний потік виводиться найпопулярніше число. У випадку, коли їх декілька - вивести всі їх за зростанням.

Приклад:

Ввід	***Вивід***
5 12 9 -19 3 7	1 9
3 22 12 2	2
2 22 33	2 3

Псевдокод:

```
ПОЧАТИ
   ОГОЛОСИТИ масив a розміром 10, заповнений нулями
   ОГОЛОСИТИ змінні n, x, max = 0
   ОТРИМАТИ n (кількість чисел)

   ДЛЯ КОЖНОГО i від 0 до n - 1
      ОТРИМАТИ x
      ПРИСВОЇТИ x абсолютне значення

      ПОКИ x не дорівнює 0
         t = x % 10 (отримати останню цифру)
         ЗБІЛЬШИТИ a[t] на 1
         ЯКЩО a[t] більше ніж max
            ПРИСВОЇТИ max=a[t]
         ПОДІЛИТИ x на 10 (відкинути останню цифру)
      КІНЕЦЬ ЦИКЛУ

   ДЛЯ КОЖНОГО i від 0 до 9
      ЯКЩО a[i] дорівнює max
         ВИВЕСТИ i
КІНЕЦЬ
```

C++ код:

```cpp
#include <iostream>
#include <cstdlib>
#include <cmath>


using namespace std;

int main()
{
  int a[10] = {0};
  int n, x;
  cin >> n;

  int max = 0;
  for(int i = 0; i < n; i++){
        cin >> x;
        x = abs(x);

```

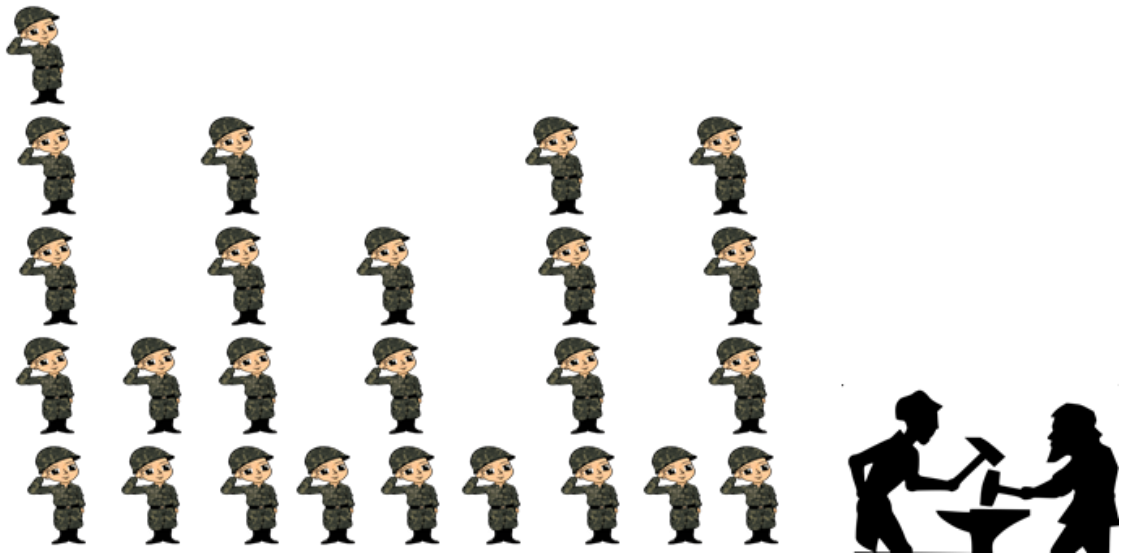
```
19.          while (x != 0){
20.            int t = x  % 10;
21.          a[t]++;
22.            if(a[t] > max){
23.              max = a[t];
24.            }
25.          x /= 10;
26.          }
27.    }
28.
29.    for(int i = 0; i < 10; i++){
30.      if(a[i] == max){
31.        cout << i << " ";
32.      }
33.    }
34.
35. }
```

Задача 5. Завдання військового програміста

(Завдання на пошук кількості парних в масиві)

Військовослужбовці стоять у ряд однією шеренгою. За кожним військовим стоїть ще ціла кількість, за кожним різна.

Керівництво військової частини хоче відібрати до тренувань з майстерності ковалів, а для цього вони слідкують за тим, щоб у кожній шерензі стояла парна кількість солдат, які кувати метал мають лише з напарником. Ті воїни, які залишаються без пари в шерензі відразу покидають стрій і йдуть писати код на комп'ютері.



Допоможіть військовому програмісту скласти програму, яка дозволятиме автоматично здійснювати замовлення кількості ковадл, вносячи до комп'ютера в одновимірну таблицю відповідну кількість солдат у шеренгах.

Технічні умови:

У першому рядку вхідного потоку записано кількість чисел N ($0 < N \leq 10000$), яка показує скільки солдат у шерензі вишикувані для відбору; в наступному рядку записано самі числа через пропуск, які не перевищують за модулем 10000 та показують скільки солдат стоїть в ряду один за одним.

Вихідні дані:

У вихідний потік вивести одне число - кількість ковадл.

Приклад:

Ввід	***Вивід***
3 3 5 7	6
5 2 2 4 8 1	8

```
1. #include <iostream>
2.
3. using namespace std;
4.
```

```
 5. int main()
 6. {
 7.
 8.   int a[10000];
 9.   int n;
10.
11.   cin >> n;
12.
13.   int result = 0;
14.   for(int i = 0; i < n; i++){
15.     cin >> a[i];
16.     result += a[i] / 2 * 2;
17.   }
18.
19.   result /= 2;
20.
21.   cout << result;
22.
23. }
24.
```

Гарне рішення, чи не так? Але навіть його можна покращити та оптимізувати, що на олімпіадах дуже потрібно. Правда ж, немає сенсу з масиву, якщо його елемент ми в циклі зчитуємо й там же використовуємо? Звісно! То можна його прибрати, а також трішки спростити логіку обчислень і елегантне та оптимізоване рішення у вас у руках!

```
 1. #include <iostream>
 2.
 3. using namespace std;
 4.
 5. int main(){
 6.   int n, temp;
 7.   cin >> n;
 8.
 9.   int result = 0;
10.   for(int i = 0; i < n; i++){
11.     cin >> temp;
12.     result += temp / 2;
13.   }
```

```
14.    cout << result;
15. }
```

Задача 6. Розрахунок квадратів для агрономів

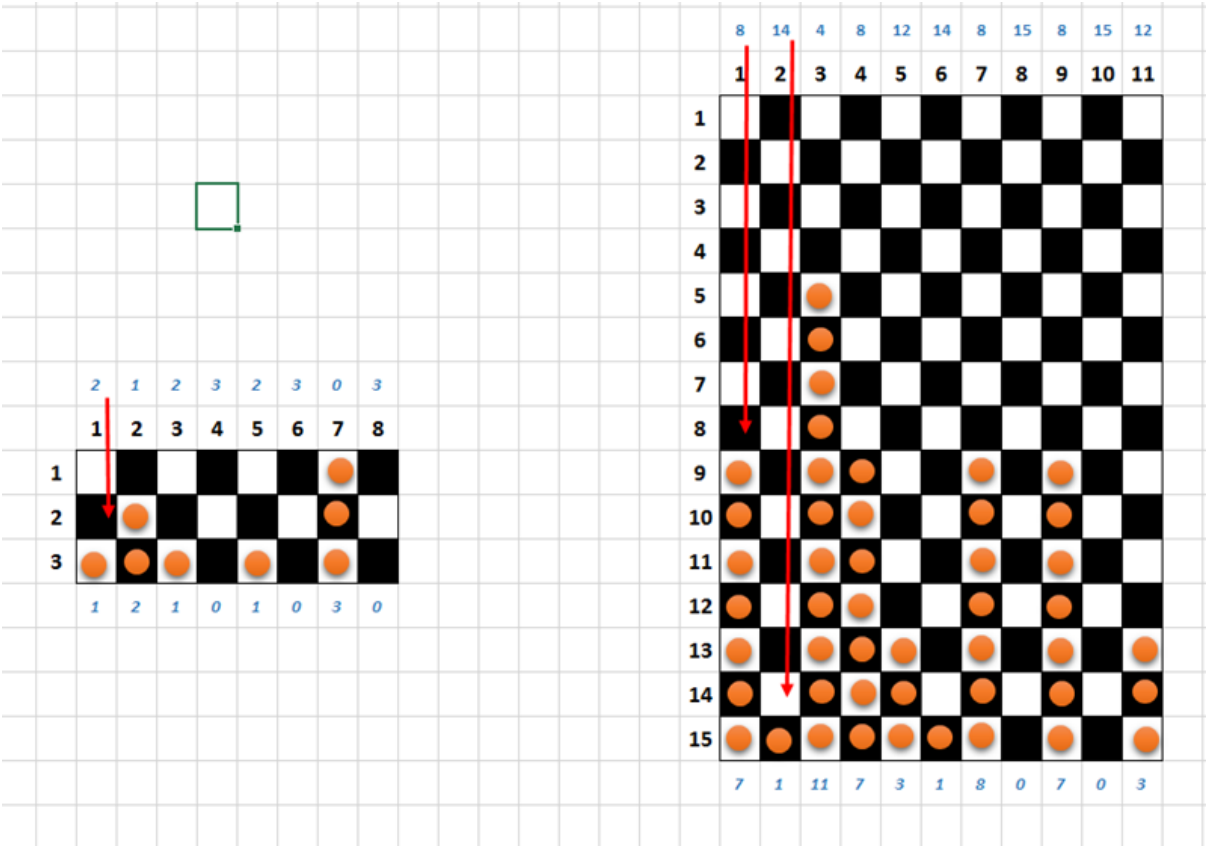
У фермерського господарства є поле прямокутної форми розміром (0 < N, M < 1000). Для зручності контролю за роботою блоку техніки агрономи розбили поле на маленькі одиничні квадрати.

Техніка може їздити лише по вертикальним клітинкам, заїжджаючи на поле з голови і до хвоста поля, тому засівання здійснюється з протилежного боку поля, щоб не топтати посіяне (на рисунку це кружечки в квадратиках).

Допоможіть написати програму, яка дозволятиме показувати загальну кількість квадратів, яку можна засіяти (кількість незайнятих кружечками квадратів). Та виводить кількість вільних горизонтальних ліній (ті, які повністю не засіяні).

Приклад:

Ввід	***Вивід***
8 3 1 2 1 0 1 0 3 0	16 0
11 15 7 1 11 7 3 1 8 0 7 0 3	117 4




```cpp
#include <iostream>

using namespace std;

int main()
{
  int n, m;
  cin >> n >> m;

  int a[1000];
  for(int i = 0; i < n; i++){
    cin >> a[i];
  }

  int max = 0;
  int sum = 0;
  for(int i = 0; i < n; i++){
    sum += a[i];
    if(a[i] > max){
      max = a[i];
    }
  }

```

```
24.   cout << n * m - sum << " " << m - max;
25. }
```

І знову ж, яким би гарним це рішення не було, але навіть його можна переробити без масивів. Правда ж, ми використовуємо масив лише 1 раз при зчитуванні?

```
 1. #include <iostream>
 2.
 3. using namespace std;
 4.
 5. int main()
 6. {
 7.   int n, m, temp;
 8.   cin >> n >> m;
 9.
10.   int longest = 0;
11.   int free = m * n;
12.   for(int i = 0; i < n; i++){
13.     cin >> temp;
14.     free -= temp;
15.     if(temp > longest){
16.       longest = temp;
17.     }
18.   }
19.
20.   cout << free << " " << m - longest;
21. }
```

7. Текстові рядки

Рядкові величини в C++ - досить цікава тема, витоки якої лежать ще в далекому минулому. Зараз рядки в цій мові можуть бути представлені як і масив символів, так і типом string.

Подання стрічок як масивів символів C++ успадкував від свого предка - мови програмування C. Але щоб відрізнити рядок від звичайного масиву в кінці нього має знаходитися '\0' (нуль-символ), який сигналізує про закінчення.

Не важко зрозуміти, щоб записати цей нуль-символ треба виділити на 1 елемент у масиві більше, тому якщо ви хочете зберегти слово "Програма" (8 символів) як рядок, то потрібно виділити на 1 символ більше: char slovo[9].

Приклади оголошення та ініціалізації рядків.

```
1. const char text[] = "зберігання символів речення"; // даємо компілятору змогу самому визначити довжину стрічки
2. char slovo[9] = "programa"; // вказуємо фактичну довжину (враховуючи нуль-символ)
3. char alphabet[] = {'A', 'B', 'C', 'D', 'E', 'R', 'I', 'K', 'L', 'm', '\0'}; // обов'язково вказуємо нуль-символ в кінці
4. char slovo[25] = "I am, Happy"; // виділяємо елементів масиву більше ніж стрічка щоб мати змогу змінити її на довшу
5.
6. char b[15]; // просто масив символів (поки не з'явиться нуль-символ в ньому)
```

Проте C++, на відміну від свого попередника, дає можливість працювати зі стрічками, які змінними типу string. Такий підхід є загальноприйнятим, зрозумілішим та більш гнучким.

```
1. string text = "зберігання символів речення"; // створюємо змінну рядкового типу
```

Приклад використання рядків:

```
1. cin >> text; // зчитування стрічки
```

```
 2.
 3. cin.get(str,80); cin.get(); cin.get(a,32);
 4. cin.getline(str, 50,'\n')
 5. //зчитування припиняється у випадку коли є символ обмеження
'\n'
 6. //або якщо є вказівник кінця файлу
 7. //або якщо кількість введених символів досягла останнього (в
нашому прикладі 50-1)
 8.
 9. cout << text; //виведення стрічки в консоль
```

Корисні функції для роботи з рядками:

- strlen(s) - визначає кількість символів в рядку (використовується у виразах);
- strcat(s1,s2) - з'єднує рядки s1 і s2, результат записує в s1;
- strncat(s1,s2,n) - до змінної s1 додає перших n символів рядка s2. (є командою);
- strcmp(s1,s2) - порівнює рядки. Повертає 0, якщо рядки рівні;
- strcpy(s1,s2) - копіює символи з рядка s2 в рядок s1;
- strncpy(s1,s2,n) - копіює перших n символів рядка s2 в рядок s1;
- strchr(s1,'a') - визначає перше входження символу у рядок s1 так: повертає рядок, який починається від першого входження заданого символу до кінця рядка s1;
- strrchr(s1,'a') - визначає останнє входження символу у рядок s1;
- strspn(s1,s2) - визначає номер першого символу, який входить у рядок s1, але не входить у рядок s2;
- strtok(s1,s2) - визначає частину рядка s1, яка закінчується перед першим однаковим символом рядків s1 та s2;
- strset(s1,'a', n) - вставляє n разів заданий символ перед рядком s1;
- strupr(s1) - перетворює усі малі літери рядка у великі;
- strlwr(s1) - перетворює усі великі літери рядка у малі.

У більшості завдань зручно використовувати символьний тип char [ ] та клас string.

Наводимо деякі корисні функції.

Значення	Функція
Довжина рядка	`s.length(); або s.size();`
Додавання рядків	`s1+s2`
Очищення рядка	`s.clear();`
Вставлення рядка	`s.insert(p,s1)`
Очищення рядка	`s.erase();`
Обмін рядків	`s.swap(s1)`
Перевіряє чи пустий рядок	`s.empty();`
Копіюємо k символів починаючи з позиції pos	`s.substr(pos, k)`
Пошук першого входження стрічки в іншу	`s.find(s2)`

Практичні завдання

Задача 1. Прізвище, ім’я та по-батькові задом наперед

Напишіть програму, яка буде виводить на екран у стандартний потік прізвище, ім’я та по-батькові задом наперед.

Приклад:

Ввід	***Вивід***
Ivanov Ivan Ivanovich	hcivonavI navI VonavI

```
1. #include <iostream>
```

```
 2. #include <string>
 3.
 4. using namespace std;
 5.
 6. int main()
 7. {
 8.    string name;
 9.    getline(cin, name); // функція використовується щоб мати
можливість зчитати стрічку з пробілами
10.
11.    // name.length() використовується щоб визначити довжину
стрічки
12.    for(int i = name.length() - 1; i >= 0; i--){ // цикл від
останнього до першого індекса масива
13.       cout << name[i];
14.    }
15.
16. }
```

Задача 2. Виграє найбільше ім’я

Реалізуйте алгоритм, який зчитуватиме зі стандартного потоку ціле число, кількість гравців, завдання яких по черзі ввести своє прізвище та ім’я з клавіатури.

Програма має виводити того, в кого воно буде найдовшим.

Передбачити ігнорування введення більше ніж одного відступу між символами.

У випадку, коли в декількох учасників гри буде однакова кількість, то виводити інформацію про них усіх. Прізвища сортувати за алфавітом.

Технічні умови:

У першому рядку вхідного потоку записано кількість учасників, які показуватимуть скільки прізвищ буде введено N ($0 < N \leq 1000$), в наступному рядку записано самі прізвища та ім’я в стовпець.

Вихідні дані:

Вивести у вихідний потік лише імена переможців.

Приклад:

Ввід	***Вивід***
3 Карявка Сергій Паламарюк Максим Коваленко Едуард	Максим Едуард

```
 1. #include <iostream>
 2. #include <string>
 3.
 4. using namespace std;
 5.
 6. int main()
 7. {
 8.   int n;
 9.   cin >> n;
10.
11.   string names[1000];
12.   int max = 0;
13.   for(int i = 0; i < n; i++){
14.     getline(cin, names[i]);
15.     int len = names[i].length();
16.     if(len > max){
17.       max = len;
18.     }
19.   }
20.
21.   for(int i = 0; i < n; i++){
22.     int len = names[i].length();
23.     if(len == max){
24.       cout << names[i].substr(names[i].find(" ") + 1) <<
endl;
25.     }
26.   }
27. }
```

Задача 3. Шифр Цезаря

Сергій пересилає Максиму секретне повідомлення через соціальні мережі, але не довіряє технологіям, тому додатково шифрує його текст відомим Шифром Цезаря.

У цьому шифрі кожна літера в повідомленні замінюється на k-ту літеру після неї в алфавіті (при k=3, літера “a” буде замінена на “г”).

Крок зміщення літери в шифрі водиться через потік на початку повідомлення через пропуск.

Приклад:

Ввід	***Вивід***
3 i know everything	в дфз їрвб

```
#include <iostream>
#include <string>

using namespace std;

int main()
{
  int key; // ключ шифру
  cin >> key;

  string text; // текст
  getchar(); // функція, яка зчитує один символ з стандартного потоку (потрібна для коректної роботи наступної getline)
  getline(cin, text); //зчитування стрічки тексту

  for(int i = 0; i < text.length(); i++) { // проходимося в циклі по кожному символу стрічки
    char ch = text[i];
    if (ch >= 'a' && ch <= 'z'){ // якщо символ між a та z
      ch = ch + key; // додаємо до символа ключ (*)
      if (ch > 'z') { // якщо символ вийшов за межі алфавіту
          ch = ch - 'z' + 'a' - 1; // переносимо його на початок
```

```
21.         }
22.         text[i] = ch; // заміняємо символ в тексті
23.       } else if (ch >= 'A' && ch <= 'Z'){ // такі ж самі
операції проводимо з великими літерами алфавіту
24.         ch = ch + key;
25.         if (ch > 'Z'){
26.             ch = ch - 'Z' + 'A' - 1;
27.         }
28.         text[i] = ch;
29.       }
30.    }
31.
32.    // (*) Примітка: в C++ символи мають своє числове
представлення, тому якщо додати до символа 'a' число 1, то
отримаємо символ 'b'.
33.
34.    cout << text;
35. }
```

Задача 4. Підрахунок сніжинок

Діду Морозу учні написали листа. Але під час вивчення в школі текстового редактора замість клавіші пробіл, вони ставили сніжинки та решітки, хто як хотів і скільки хотів.

Допоможіть знайти всю кількість сніжинок та решіток у короткому посланні.

Приклад:

Ввід	***Вивід***
Хочу*щоб*всі**були*живі**й**здорові	9
хочемо#*снігу***багато#і*радості*навколо###	11

```
 1. #include <iostream>
 2. #include <string>
 3.
 4. using namespace std;
 5.
 6. int main()
 7. {
 8.   string text;
 9.   getline(cin, text);
10.
11.   int count = 0;
12.   for(int i = 0; i < text.length(); i++) {
13.     if(string("*#").find(text[i]) != -1){ // пошук в стрічці
"*#" символа text[i] передбачає, що якщо символа немає, то
функція повертає значення -1
14.       count++;
15.     }
16.   }
17.
18.   cout << count;
19. }
```

8. Робота з файлами

Досить часто олімпіадні задачі пропонують виконати завдання, пов’язані із зчитуванням даних, розміщених у файлі.

Серед завдань зустрічаються: організація роботи з великими обсягами даних, які подали у файлі, зчитування даних з файлів, перезапис наявних даних, формування відповідних вихідних файлів тощо.

Для роботи з файлами підключається відповідна бібліотека

```
#include <fstream>
```

Під час роботи з файлами слід дотримуватись певних правил та рекомендацій, уважно вивчати умови завдань. Завжди варто звернути увагу на назву, яку вам пропонує задача, і яку назву використовуєте ви у своєму алгоритмі.

Наприклад:

Вхідний файл - файл, з якого зчитуються вхідні дані: input.txt

Вихідний файл - файл, у який записуються результати виконання програми: output.txt

Уважно слідкуйте за форматом вхідних та вихідних даних, з якими працює алгоритм.

Для об'ємних наборів даних слід використовувати ефективні структури даних (наприклад, вектори або масиви), щоб уникнути втрати даних або переповнення виділеної пам’яті.

Задачі, які пропонуються під час олімпіад з програмування, часто вимагають жорсткі обмеження по часу виконання та використанню пам'яті. Тому важливо використовувати ефективні алгоритми і правильно організовувати роботу з файлами.

Відповіді на рішення приймає зазвичай автоматична система, яка вимагає роботи з еталонними файлами. Тобто маємо справу з чітким контролем та структурою вихідного файлу. Звертайте увагу на всі дрібниці, крапки, коми, відступи, кількість рядків тощо.

Завжди аналізуйте завдання з глибоким розумінням того, про що саме йде мова в задачі, і який результат має подаватись на перевірку.

Алгоритм роботи з файлами.

1. Підготовка файлу.
   - Заздалегідь створіть або підготуйте файли з вхідними даними та місце для вихідних результатів або переконайтеся, що ви працюєте з тим файлом, який пропонується в задачі.
   - Вхідний файл (наприклад, input.txt) повинен містити дані у форматі, описаному в умові задачі.
   - Вихідний файл (наприклад, output.txt) створюється програмою для збереження результатів. Слідкуйте, яким чином у нього внесено дані.
2. Зчитування даних з файлу.
   - За допомогою класу ifstream відкрийте вхідний файл для читання.
   - За допомогою класу ofstream відкрийте вихідний файл для запису.
3. Зчитування даних.
   - Використовуйте оператори вводу >> або методи класу ifstream для зчитування даних із вхідного файлу. Варто детально дослідити ці механізми окремо.
   - Можливо, вам потрібно буде зчитати кілька типів даних (числа, рядки, масиви тощо). Уважно слідкуйте за форматом даних.
4. Обробка даних.
   - У цьому випадку це буде робота самого алгоритму, який має певним чином обробити, перетворити і підготувати дані до наступного етапу, а саме запису або перезапису відповідного вихідного файлу.
5. Запис або перезапис результатів у вихідний файл.
   - Після обробки даних результат потрібно записати у вихідний файл.
   - Використовуйте оператори виводу << або методи класу ofstream.
6. Закрити файли.

- Після завершення роботи обов'язково закрийте обидва файли за допомогою методу close(). Це необхідно для збереження даних та звільнення ресурсів.

7. Перевірка результатів

- Перевірте коректність формату виводу. Часто навіть одна зайва або пропущена цифра чи пробіл можуть призвести до невірного результату.

Наступні приклади демонструють певні фрагменти організації роботи з файлами.

- **Читаємо файл** (перед читанням потрібно створити файл):

```
1. #include <fstream> // підключаємо бібліотеку для роботи з
файлами
2. using namespace std;
3.
4. int main()
5. {
6.  ifstream file; // створюємо об'єкт класу ifstream
7.  file.open("d:\\1\\файл.txt"); // відкриваємо файл
8. }
```

- **Перевірка чи відкрився файл:**

```
 1. #include <fstream>
 2. using namespace std;
 3.
 4. int main()
 5. {
 6.
 7.  ifstream file ("d:\\1\\файл.txt");
 8.
 9.  if (!file)
10.  {
11.  cout << "Файл не відкритий\n\n";
12.  return -1;
13.  }
14.  else
15.  {
16.  cout << "Все ОК! Файл відкрито!\n\n";
17.  return 1;
18.  }
19. }
```

- **Зчитування з файлу:**

```
1. double d;
2. int i;
3. string s;
4.
5. file >> d >> i >> s;
```

Приклад організації роботи з файлом:

Файл містить до 50 натуральних чисел, розділених між собою через пробіл. Потрібно переписати числа в зворотному порядку. У даному випадку слід пам'ятати, що ви маєте підготовлений вхідний файл і відповідно вихідний файл.

```
 1. #include <iostream>
 2. #include <fstream>
 3.
 4. int main() {
 5.     // Відкриваємо файл "input.txt" для читання даних
 6.     std::ifstream fin("input.txt");
 7.
 8.     // Перевіряємо, чи файл вдалося відкрити
 9.     if (!fin) {
10.         std::cerr << "Не вдалося відкрити файл input.txt" <<
std::endl;
11.         return 1;
12.     }
13.
14.     // Створюємо статичний масив для зберігання максимум 50
чисел
15.     const int SIZE = 50;
16.     int numbers[SIZE];
17.     int count = 0; // Лічильник фактичної кількості чисел
18.
19.     // Читаємо числа з файлу і записуємо їх у масив, але не
більше 50 чисел
20.     while (fin >> numbers[count] && count < SIZE) {
21.         ++count;
22.     }
23.
24.     // Закриваємо вхідний файл
25.     fin.close();
```

```
26.
27.     // Відкриваємо файл "output.txt" для запису даних
28.     std::ofstream fout("output.txt");
29.
30.     // Переписуємо числа у зворотному порядку
31.     for (int i = count - 1; i >= 0; --i) {
32.         fout << numbers[i] << " ";
33.     }
34.
35.     // Закриваємо вихідний файл після завершення запису
36.     fout.close();
37.
38.     return 0;
39. }
```

Практичні завдання

Задача 1. Файл з таблицею множення

Вивести таблицю множення у файл.

```
 1. #include <iostream>
 2. #include <fstream> //бібліотека для роботи з файлами
 3.
 4. using namespace std;
 5.
 6. int main() {
 7.   ofstream output("output.txt"); //відкриваємо файл для
запису результату
 8.
 9.   for(int i = 1; i <= 10; i++){ //в циклі по першому
множнику (від 1 до 10)
10.     for(int j = 1; j <= 10; j++){ //в циклі по другому
множнику (від 1 до 10)
11.       output << i << " * " << j << " = " << i * j << endl;
12.     }
13.   }
14.
15.   output.close(); // закриваємо файл
16. }
```

Задача 2. Сортування у файлі

Написати програму, зчитує з файлу input.txt числа та знаходить суму максимального та мінімального числа. Результат записати у вихідний файл output.txt.

Технічні умови:

З файлу input.txt зчитується натуральне число N - кількість чисел, яке не перевищує 100. Після цього зчитується N чисел.

Вихідні дані:

У файл output.txt програма має записати математичний вираз: <*мінімальне число*> + <*максимальне число*> = <*сума*>

Приклад:

Ввід	***Вивід***
5 -3 5 2 -4 1	-4 + 5 = 1

```
 1. #include <iostream>
 2. #include <fstream>
 3.
 4. int main() {
 5.     // Відкриваємо файл "input.txt" для читання даних
 6.     std::ifstream fin("input.txt");
 7.
 8.     // Перевіряємо, чи файл вдалося відкрити
 9.     if (!fin) {
10.         std::cerr << "Не вдалося відкрити файл input.txt" <<
std::endl;
11.         return 1;
12.     }
13.
14.     // Зчитуємо кількість чисел N
15.     int N;
16.     fin >> N;
17.
18.     // Перевіряємо, чи кількість чисел не перевищує 100
19.     if (N <= 0 || N > 100) {
```

```
20.         std::cerr << "Невірна кількість чисел у файлі" <<
std::endl;
21.         return 1;
22.     }
23.
24.     // Статичний масив для зберігання N чисел
25.     int numbers[100];
26.
27.     // Читаємо перше число, щоб ініціалізувати мінімальне і
максимальне значення
28.     fin >> numbers[0];
29.     int minNumber = numbers[0];
30.     int maxNumber = numbers[0];
31.
32.     // Читаємо інші числа з файлу і знаходимо мінімальне та
максимальне
33.     for (int i = 1; i < N; ++i) {
34.         fin >> numbers[i];
35.         if (numbers[i] < minNumber) {
36.             minNumber = numbers[i];
37.         }
38.         if (numbers[i] > maxNumber) {
39.             maxNumber = numbers[i];
40.         }
41.     }
42.
43.     // Закриваємо вхідний файл
44.     fin.close();
45.
46.     // Відкриваємо файл "output.txt" для запису результату
47.     std::ofstream fout("output.txt");
48.
49.     // Обчислюємо суму мінімального і максимального числа
50.     int sum = minNumber + maxNumber;
51.
52.     // Записуємо результат у вигляді математичного виразу
53.     fout << minNumber << " + " << maxNumber << " = " << sum
<< std::endl;
54.
55.     // Закриваємо вихідний файл після завершення запису
56.     fout.close();
57.
58.     return 0;
59. }
```

9. Структури даних

Структура даних - це спосіб представлення інформації. Можна сказати, що структури даних є своєрідним інструментом проєктування алгоритмів. Основним підбором виступає принцип збереження даних у пам'яті комп'ютера та з організацією читання збереженої інформації, що структура, яка містить набір певних елементів стала, так би мовити єдиним цілим, і з нею зручно було працювати.

У програмуванні застосовуються наступні типи структур:

- **Масив** - впорядкований набір елементів.
- **Множина** - набір унікальних елементів
- **Рядок** - так-так, ті звичайні рядки - string.
- **Список** - як ланцюжок, кожен попередній елемент посилається на наступний.
- **Мапа** - схоже на таблицю, де є набір ключа та значення.
- **Черга** - як в магазині: перший прийшов - перший пішов.
- **Стек** - як купа книжок: щоб дістати нижню треба вийняти всі верхні.
- **Граф** - як дороги між містами.
- **Дерево** - схоже сімейне дерево.
- **Дек** - та неприємна черга в магазині, коли людина може стати, як на початок, так і в кінець.

Реалізація деяких класичних алгоритмів пропонується в наступних прикладах.

10. Стандартна бібліотека шаблонів

Вектор (vector)

Контейнер vector можна представити, як обгортку над звичайним масивом, яка полегшує роботу з ним. Найбільша його перевага - динамічність розміру, тобто здатність розширюватися, коли це потрібно.

Створити об’єкт вектора можна наступним чином:

```
1. /* Загальний шаблон:
2. vector<type> name; ,
3. де type - це тип даних об’єктів вектора, а name - назва
вектора
4. */
5.
6. vector<int> a; // створюється пустий вектор з вмістимістю 0
7. vector<int> b = {1, 2, 3}; //вектор з розміром 3 та
елементами {1, 2, 3}
8. vector<int> c(10) //вектор з розміром 10, заповнений нулями
```

Звертатися до елементів вектора можна так само, як до елементів масива:

```
1. vector<int> b = {1, 2, 3};
2.
3. cout << b[1] << endl; // 2
4. b[1] = 25;
5. cout << b[1] << endl; // 25
```

Деякі корисні методи для роботи з vector:

`push_back(value)`
 	
- додавання елементу в кінець вектора
	`1. vector<int> v;  // v = {}`
	`2. v.push_back(1); // v = {1}`
	`3. v.push_back(2); // v = {1, 2}`
	`4. v.push_back(3); // v = {1, 2, 3}`

`clear()`
 	
- очищення вектора
	`1. vector<int> v = {1, 2, 3}; // v = {1, 2, 3}`
	`2. v.clear(); // v = {}`

`pop_back()`
 	
- видалення останнього елементу вектора
	1. vector<int> v = {1, 2, 3, 4}; // v = {1, 2, 3, 4}
	2. v.pop_back(); // v = {1, 2, 3}

`size()`
 	
- визначення розміру вектора
	1. vector<int> v = {1, 2, 3};
	2. v.size(); // 3
	3. v.pop_back();
	4. v.size(); // 2

`capacity()`
 	
- визначення вмістимості вектора
	1. vector<int> v = {1, 2, 3};
	2. v.capacity(); // 3
	3. v.pop_back();
	4. v.capacity(); // 3

`empty()`
 	
- перевірка чи вектор пустий
	1. vector<int> v = {1, 2, 3};
	2. v.empty(); // false
	3.
	4. vector<int> u;
	5. v.empty(); // true

`resize(n)`
 	
– зміна розміру вектора
	1. vector<int> v = {5, 2, 3};
	2. v.resize(5); // v = {5, 2, 3, 0, 0}
	3. v.resize(1); // v = {5}

Множина (set)

Контейнер set у C++ — це асоціативний контейнер, що зберігає унікальні елементи в упорядкованому порядку. Кожен елемент з'являється тільки один раз і автоматично сортується в контейнері. Це корисно, коли потрібно гарантувати відсутність дублікатів у наборі даних, а також мати можливість швидкого пошуку або перевірки наявності елементів. Наприклад, у реальному житті можна використовувати set для зберігання унікальних номерів квитків на концерт, де кожен номер має бути присутній лише один раз.

```cpp
#include <iostream>
#include <set>

using namespace std;

int main() {
    set<int> ticketNumbers;

    // Додавання елементів
    ticketNumbers.insert(101);
    ticketNumbers.insert(202);
    ticketNumbers.insert(303);

    // Спроба додати дублікат
    ticketNumbers.insert(101);

    // Виведення всіх номерів квитків
    cout << "Усі унікальні номери квитків: ";
    for (int n : ticketNumbers) {
        cout << n << " "; // результат - 101 202 303
    }

    return 0;
}
```

Мапа (map)

Контейнер map у C++ є асоціативним контейнером, що зберігає пари "ключ-значення", причому кожен ключ унікальний, а значення може повторюватися. Основною особливістю map є те, що він автоматично підтримує сортування елементів за ключами, що робить його зручним для швидкого пошуку, вставки та видалення елементів. У реальному житті прикладом може бути телефонний довідник, де ключем є ім'я людини, а значенням — її номер телефону. Цей контейнер ідеально підходить для ситуацій, коли потрібно швидко знаходити дані за унікальним ідентифікатором (ключем), наприклад, у словниках або базах даних.

```cpp
#include <iostream>
#include <map>
#include <string>

using namespace std;

int main() {
    map<string, string> phoneBook;

    // Додавання елементів
    phoneBook["Аліса"] = "123-456-789";
    phoneBook["Іван"] = "987-654-321";

    // Пошук номера телефону за іменем
    string name = "Аліса";
    if (phoneBook.find(name) != phoneBook.end()) {
        cout << "Номер телефону " << name << ": " <<
phoneBook[name] << endl;
    } else {
        cout << name << " не знайдено в телефонній книзі."
<< endl;
    }

    return 0;
}
```

11. Вступ до алгоритмізації

Досить широка і потужна тема, яка виходить за межі цього збірника і потребує більш серйозної уваги та може стати наступним кроком для дослідження.

Сутність використання алгоритмів в олімпіадному програмуванні полягає в їх універсальності, ефективності та здатності вирішувати широкий спектр задач у мінімальні строки з мінімальними ресурсами. Класичні алгоритми є фундаментальними інструментами для розв'язання типових проблем, з якими стикаються учасники олімпіад. Їхнє застосування дозволяє досягти оптимальних результатів і уникнути повторного "винайдення колеса".

Прикладом таких алгоритмів є алгоритми пошуку найкоротшого шляху (алгоритм Дейкстри), пошук (бінарний пошук, пошук стрічки в тексті), сортування (сортування злиттям, швидке сортування) або ж ефективні алгоритми роботи з графами (алгоритм Крускала для знаходження мінімального кістяка).

Класичні алгоритми забезпечують програмістів шаблонами розв'язання типових проблем. Вивчаючи ці алгоритми, учасники олімпіад можуть швидко розпізнавати певні структури задач та підібрати відповідні алгоритмічні рішення. Наприклад, задачі на графи часто розв'язуються за допомогою BFS (пошук у ширину), DFS (пошук у глибину), алгоритмів для знаходження компонент зв'язності або циклів.

Пошукові алгоритми

Пошукові алгоритми — це методи, які використовуються для знаходження конкретного елемента або множини елементів у структурі даних, таких як масиви, списки чи графи. Вони включають різні підходи, зокрема лінійний пошук, бінарний пошук, алгоритми пошуку на графах (як-от алгоритм Дейкстри чи пошук у глибину) та інші.

Пошук перебором

Пошук перебором або лінійний пошук є одним із найпростіших алгоритмів пошуку в програмуванні. Основна ідея методу перебрати послідовно кожен елемент масиву або іншої структури даних, поки не знайдеться потрібний елемент або не будуть переглянуті всі елементи.

Використовуючи псевдокод, можна продемонструвати загальну ідею лінійного пошуку:

```
1. ПОЧАТИ ФУНКЦІЮ linearSearch(arr[], size, target)
2. ДЛЯ КОЖНОГО i від 0 до size - 1
3. ЯКЩО arr[i] дорівнює target
4. ПОВЕРНУТИ i
5. ПОВЕРНУТИ -1  // якщо тут, то елемент відсутній
6. КІНЕЦЬ ФУНКЦІЇ
```

Під час виконання лінійного пошуку не має значення чи відсортований масив, тому основна ідея зводиться до покрокової перевірки кожного елемента, до поставленого критерія з допомогою циклічного перебору кожного елемента.

Наприклад:

```
 1. #include <iostream>
 2. using namespace std;
 3.
 4. int main() {
 5.     // Ініціалізуємо масив
 6.     int arr[] = {10, 20, 30, 40, 50};
 7.     int size = sizeof(arr) / sizeof(arr[0]);
 8.
 9.     // Елемент, який шукаємо
10.     int target = 30;
11.     bool found = false;
12.
13.     // Лінійний пошук
14.     for (int i = 0; i < size; i++) {
15.         if (arr[i] == target) {
16.             cout << "Елемент " << target << " знайдено на
позиції: " << i << endl;
17.             found = true;
```

```
18.             break;  // Вихід з циклу, якщо елемент знайдено
19.         }
20.     }
21.
22.     // Якщо елемент не знайдено
23.     if (!found) {
24.         cout << "Елемент " << target << " не знайдено" <<
endl;
25.     }
26.
27.     return 0;
28. }
```

Задача 1. Лінійний пошук стрічки

Максим шукає у великому тексті улюблене слово, допоможіть йому знайти позицію цього числа в тексті.

Якщо слово є в реченні - вивести позицію початку, а інакше - вивести “No”.

Приклад:

Ввід	***Вивід***
Hello world! world	6
Hello world! top	NO!

Псевдокод:

```
 1. ПОЧАТИ ФУНКЦІЮ find(text, sub)
 2.    ДЛЯ КОЖНОГО i від 0 до довжини text - 1
 3.       ВСТАНОВИТИ present = true
 4.       ДЛЯ КОЖНОГО j від 0 до довжини sub - 1
 5.          ЯКЩО символ text[i + j] не дорівнює sub[j]
 6.             ВСТАНОВИТИ present = false
 7.       КІНЕЦЬ ВНУТРІШНЬОГО ЦИКЛУ
 8.       ЯКЩО present дорівнює true
 9.          ПОВЕРНУТИ i
10.    КІНЕЦЬ ЗОВНІШНЬОГО ЦИКЛУ
```

```
11.    ПОВЕРНУТИ -1
12. КІНЕЦЬ ФУНКЦІЇ
```

C++ код:

```
 1. #include <iostream>
 2. #include <vector>
 3.
 4. using namespace std;
 5.
 6. int find(string text, string sub){
 7.   for(int i = 0; i < text.length(); i++){
 8.     bool present = true;
 9.     for(int j = 0; j < sub.length(); j++){
10.       if(text[i + j] != sub[j]){
11.         present = false;
12.       }
13.     }
14.     if(present){
15.       return i;
16.     }
17.   }
18.   return -1;
19. }
20.
21. int main() {
22.   string text;
23.   getline(cin, text);
24.
25.   string substring;
26.   getline(cin, substring);
27.
28.   int position = find(text, substring);
29.   if(position == -1){
30.     cout << "NO!";
31.   } else {
32.     cout << position;
33.   }
34. }
```

Бінарний пошук

Основна ідея **бінарного пошуку** полягає в тому, щоб скоротити кількість елементів, які потрібно перевірити, поділяючи відсортований масив на дві частини на кожному кроці. Це дозволяє значно прискорити пошук порівняно з лінійним перебором.

У цьому випадку потрібно обов'язково відсортувати елементи масиву.

Покроковий алгоритм організації бінарного пошуку може мати наступний вигляд:

1) відсортувати елементи масиву, у якому буде здійснюватись пошук;
2) знайти елемент масиву, який розміщено посередині;
3) якщо середній елемент відповідає шуканому - пошук завершується;
4) якщо середній елемент більший за шукане значення, пошук продовжується в лівій частині масиву за подібною схемою;
5) якщо менший — у правій частині.

Псевдокод:

```
 1. ПОЧАТИ
 2. ВІДКРИТИ файл input.txt для читання
 3. ВІДКРИТИ файл output.txt для запису
 4. ЗЧИТАТИ кількість елементів (size)
 5. СТВОРИТИ порожній вектор numbers для зберігання чисел
 6. ПОВТОРИТИ size РАЗІВ
 7. ЗЧИТАТИ число
 8. ДОДАТИ число у вектор numbers
 9. ЗЧИТАТИ шукане число (number)
10. ОГОЛОСИТИ змінні:
11. l = 0 (ліва межа)
12. r = size - 1 (права межа)
13. m = середина між l та r
14. result = false (початкове значення результату пошуку)
15. ПОКИ l < r - 1 і result == false
16. ЯКЩО число на середині (numbers[m]) дорівнює шуканому числу
17. result = true (знайдено число)
18. ІНАКШЕ ЯКЩО numbers[m] менше шуканого числа
19. ОНОВИТИ ліву межу (l = m)
```

```
20. ІНАКШЕ
21. ОНОВИТИ праву межу (r = m)
22. ПЕРЕРАХУВАТИ середину (m = (l + r) / 2)
23. ЯКЩО результат пошуку true
24. ВИВЕСТИ "YES" у файл output.txt
25. ІНАКШЕ
26. ВИВЕСТИ "NO" у файл output.txt
27.
28. ЗАКРИТИ файл input.txt
29. ЗАКРИТИ файл output.txt
30. КІНЕЦЬ
```

C++ код:

```
 1. #include <iostream>
 2. #include <fstream> //бібліотека для роботи з файлами
 3. #include <bits/stdc++.h> //бібліотека для структур даних
(тут використовується вектор) і алгоритмів (сортування)
 4.
 5. using namespace std;
 6.
 7. int main() {
 8.   ifstream input("input.txt"); //відкриваємо вхідний файл на
читання
 9.   ofstream output("output.txt"); //відкриваємо файл для
запису результату
10.
11.   int size; //змінна для кількості елементів
12.   input >> size; //зчитуємо число (кількість елементів) з
файлу
13.
14.   vector<int> numbers; // вектор чисел (те саме що масив,
але дає класний метод push_back, який додає елемент в кінець
вектору. це значить, що нам не треба думати про розмір вектору і
він сам може збільшуватися коли треба)
15.   int number; //змінна для зчитування чисел
16.   for(int i = 0; i < size; i++){ //в циклі size разів
17.     input >> number; //зчитуємо число з файлу
18.     numbers.push_back(number); //закидаємо число у вектор
19.   }
20.
21.   input >> number; //зчитуємо число яке будемо шукати
22.
```

```
23.   // Примітка: бінарний пошук працює ЛИШЕ з посортованим
масивом
24.
25.   int l = 0, r = size - 1, m = (l + r) / 2; //оголошуємо
змінні (l - ліва границя пошуку, r - права, m - середина пошуку)
26.   bool result = false; // результат пошуку (true - є число в
масиві, false - немає)
27.
28.   while(l < r - 1 && result == false){ //поки ліва границя
менше ніж права і результат пошуку - false
29.     if(numbers[m] == number){ // якщо число на середині
пошуку дорівнює шуканому
30.       result = true; //результат - true
31.     } else if (numbers[m] < number) { // інакше, якщо число
на середині менше шуканого
32.       l = m; // переносимо ліву границю на середину
33.     } else { //інакше
34.       r = m; // переносимо праву на середину
35.     }
36.     m = (l + r) / 2; //обчислюємо середину пошуку знову (бо
границі здвинулися)
37.   }
38.
39.   // виводимо результат у файл
40.   if(result == true){
41.     output << "YES";
42.   } else {
43.     output << "NO";
44.   }
45.
46.   // закриваємо файли
47.   input.close();
48.   output.close();
49. }
```

Алгоритми сортування

Алгоритми сортування — це методи впорядкування елементів у списку або масиві за певним критерієм, наприклад, за зростанням чи спаданням значень. До найвідоміших алгоритмів сортування належать сортування бульбашкою, сортування вставками, швидке сортування (quicksort) та сортування злиттям (merge sort).

Сортування бульбашкою

Використовується для **сортування** масиву, обмінюючи сусідні елементи, якщо вони стоять у неправильному порядку. Назва алгоритму і містить саму ідею сортування. Потрібний шуканий елемент під час організації перебору елементів спливає на поверхню (на перше місце в масиві).

Сам механізм можна пояснити простим прикладом, де потрібно відсортувати декілька чисел.

Наприклад:

Крок	**Масив після проходу**
Початковий стан	5 3 8 1
1-й прохід (1-й крок)	3 5 8 1
1-й прохід (2-й крок)	3 5 8 1
1-й прохід (3-й крок)	3 5 1 8
2-й прохід (1-й крок)	3 5 1 8
2-й прохід (2-й крок)	3 1 5 8
3-й прохід (1-й крок)	1 3 5 8

До цієї задачі будемо мати наступний код:

Псевдокод:

```
ПОЧАТИ ФУНКЦІЮ bubbleSort(arr[], size)
   ДЛЯ КОЖНОГО i від 0 до size - 2
      ДЛЯ КОЖНОГО j від 0 до size - i - 2
         ЯКЩО arr[j] більше, ніж arr[j + 1]
            ЗРОБИТИ ОБМІН:
               ВСТАНОВИТИ temp = arr[j]
               ВСТАНОВИТИ arr[j] = arr[j + 1]
               ВСТАНОВИТИ arr[j + 1] = temp
      КІНЕЦЬ ВНУТРІШНЬОГО ЦИКЛУ
   КІНЕЦЬ ЗОВНІШНЬОГО ЦИКЛУ
КІНЕЦЬ ФУНКЦІЇ

```

C++ код:

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

void bubbleSort(int arr[], int size) {
    // Проходимо кілька разів через масив
    for (int i = 0; i < size - 1; i++) {
        // На кожній ітерації "спливає" найбільший елемент
        for (int j = 0; j < size - i - 1; j++) {
            // Порівнюємо сусідні елементи
            if (arr[j] > arr[j + 1]) {
                // Якщо елемент ліворуч більший, міняємо їх місцями
                int temp = arr[j];
                arr[j] = arr[j + 1];
                arr[j + 1] = temp;
            }
        }
    }
}

int main() {
    int arr[] = {5, 3, 8, 1};  // Масив для сортування
    int size = sizeof(arr) / sizeof(arr[0]);

```

```
    // Викликаємо функцію сортування бульбашкою
    bubbleSort(arr, size);

    // Виводимо відсортований масив
    cout << "Відсортований масив: ";
    for (int i = 0; i < size; i++) {
        cout << arr[i] << " ";
    }

    return 0;
}
```

Сортування злиттям

Основна ідея полягає в розбитті масиву на менші підмасиви та сортуванні цих підмасивів окремо, а потім їх злитті у відсортований масив.

Алгоритм застосовується в ситуаціях, де потрібна стабільність сортування (якщо два елементи мають однакові значення, вони залишаються в тому ж порядку після сортування, як і до нього) або коли важливо забезпечити передбачувану продуктивність на великих наборах даних.

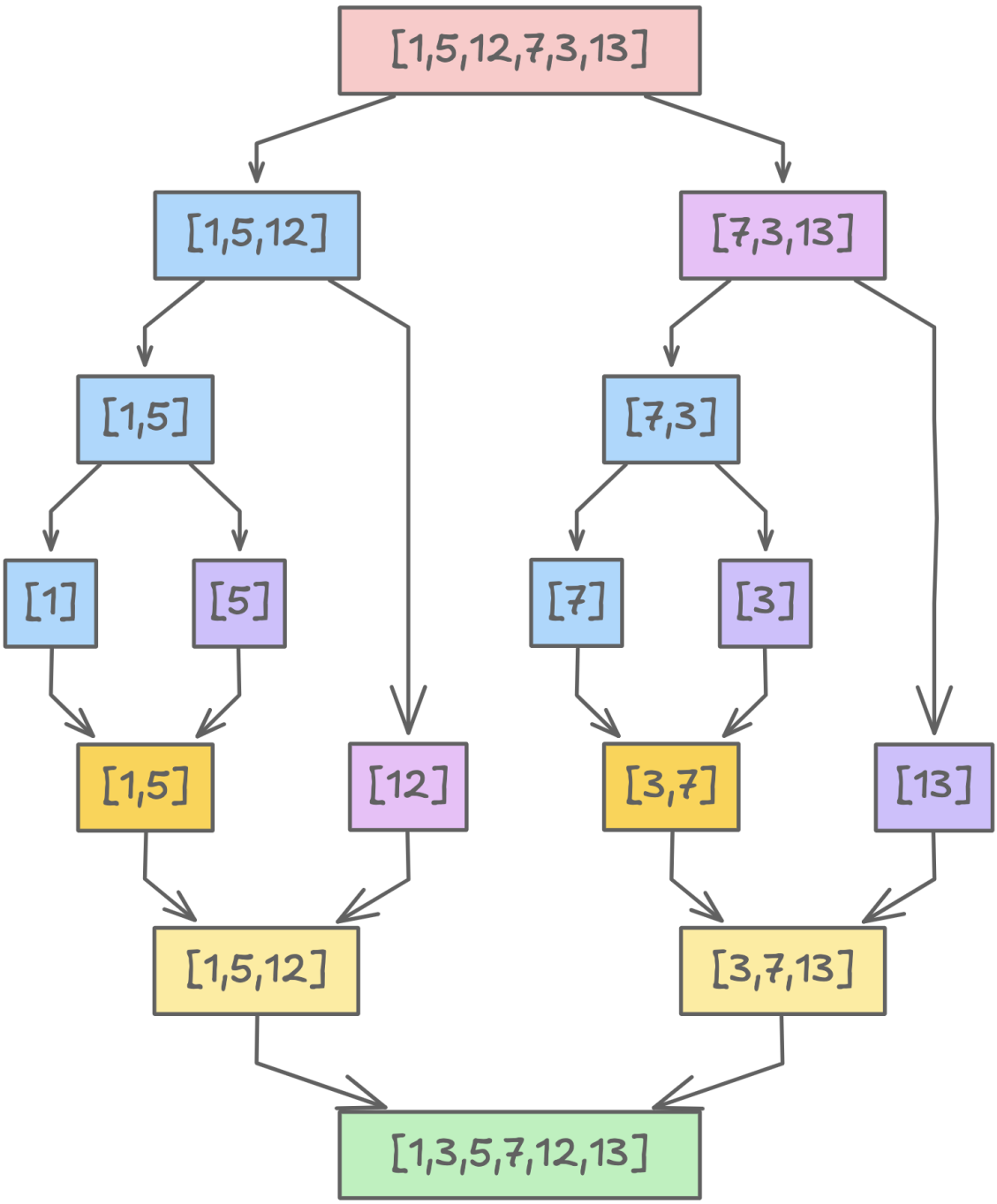
Приклад:

відсортувати масив [1,5,12,7,3,13] методом сортування злиттям.

Увесь процес організуємо в двох частинах:

- **розбиття масиву** на частини, поки не отримаємо по одному елементу;
- **злиття з сортуванням**.

Яскраво процес трансформації відображено в діаграмі на малюнку.



C++ код:

```
1. #include <iostream>
2. using namespace std;
3. 
4. // Функція для злиття двох підмасивів
5. void merge(int arr[], int left, int mid, int right) {
6.     int n1 = mid - left + 1;  // Розмір лівого підмасиву
7.     int n2 = right - mid;     // Розмір правого підмасиву
```

```
 8.
 9.     // Створюємо тимчасові масиви
10.     int leftArr[n1], rightArr[n2];
11.
12.     // Копіюємо дані в тимчасові масиви
13.     for (int i = 0; i < n1; i++)
14.         leftArr[i] = arr[left + i];
15.     for (int j = 0; j < n2; j++)
16.         rightArr[j] = arr[mid + 1 + j];
17.
18.     // Індекси для ітерації по лівому, правому підмасивам і
злиттю
19.     int i = 0, j = 0, k = left;
20.
21.     // Злиття двох підмасивів у вихідний масив
22.     while (i < n1 && j < n2) {
23.         if (leftArr[i] <= rightArr[j]) {
24.             arr[k] = leftArr[i];
25.             i++;
26.         } else {
27.             arr[k] = rightArr[j];
28.             j++;
29.         }
30.         k++;
31.     }
32.
33.     // Копіюємо залишки елементів з лівого підмасиву
34.     while (i < n1) {
35.         arr[k] = leftArr[i];
36.         i++;
37.         k++;
38.     }
39.
40.     // Копіюємо залишки елементів з правого підмасиву
41.     while (j < n2) {
42.         arr[k] = rightArr[j];
43.         j++;
44.         k++;
45.     }
46. }
47.
48. // Функція для рекурсивного сортування методом злиття
49. void mergeSort(int arr[], int left, int right) {
50.     if (left < right) {
```

```
51.         // Знаходимо середню точку
52.         int mid = left + (right - left) / 2;
53.
54.         // Рекурсивно сортуємо обидві половини
55.         mergeSort(arr, left, mid);
56.         mergeSort(arr, mid + 1, right);
57.
58.         // Зливаємо відсортовані половини
59.         merge(arr, left, mid, right);
60.     }
61. }
62.
63. int main() {
64.     // Ініціалізуємо масив
65.     int arr[] = {38, 27, 43, 3, 9, 82, 10};
66.     int size = sizeof(arr) / sizeof(arr[0]);
67.
68.     // Виводимо початковий масив
69.     cout << "Початковий масив: ";
70.     for (int i = 0; i < size; i++)
71.         cout << arr[i] << " ";
72.     cout << endl;
73.
74.     // Викликаємо алгоритм Merge Sort
75.     mergeSort(arr, 0, size - 1);
76.
77.     // Виводимо відсортований масив
78.     cout << "Відсортований масив: ";
79.     for (int i = 0; i < size; i++)
80.         cout << arr[i] << " ";
81.     cout << endl;
82.
83.     return 0;
84. }
```

У представленому коді функція `mergeSort` викликає сама себе. А як працює рекурсія?

Висновки

Збірник задач із мови програмування C++ стане в нагоді для всіх, хто бажає поглибити свої знання в цій мові та підготуватися до участі в олімпіадах з програмування. Розглянуті в збірнику завдання охоплюють широкий спектр тем, починаючи від базових концепцій і закінчуючи складними алгоритмічними задачами, які є типовими для олімпіадного програмування.

Працюючи з цим матеріалом, читач отримує можливість не лише закріпити теоретичні знання, а й відпрацювати практичні навички, що є ключовими для успішної участі в конкурсах і олімпіадах. Збірник сприяє розвитку аналітичного мислення, навичок оптимізації рішень і вміння ефективно використовувати можливості мови C++ для розв'язання складних задач.

Наша команда сподівається, що цей збірник стане надійним помічником для кожного бажаючого побачити, як задачі реалізуються з допомогою алгоритмів на C++, та стане корисним для старту у світі олімпійського програмування.

Алгоритми, написані мовою C++, часто виявляються швидшими на олімпіадах з програмування з кількох причин:

- **Низький рівень мови**

C++ є мовою низького рівня, яка надає прямий доступ до пам'яті й апаратних ресурсів комп'ютера. Завдяки цьому програми можуть працювати швидше, оскільки мінімізується кількість "посередників" між кодом і процесором. Це дозволяє оптимізувати виконання коду на рівні машинних інструкцій.

- **Відсутність автоматичного управління пам'яттю**

На відміну від деяких інших мов програмування, таких як Java чи Python, у C++ немає збирача сміття (Garbage Collector), який автоматично очищує непотрібні об'єкти з пам'яті. Хоча управління пам'яттю вручну може ускладнити процес програмування, це також знижує затримки через операції зі звільнення пам'яті, що може бути вирішальним для високоефективного коду на олімпіадах.

- **Стандартна бібліотека шаблонів (STL)**

C++ містить потужну стандартну бібліотеку шаблонів (STL), яка включає добре оптимізовані структури даних (наприклад, вектори, множини, карти) та алгоритми (сортування, пошук тощо). Ці інструменти не тільки пришвидшують розробку, але й забезпечують високу продуктивність програм.

- **Компіляція в машинний код**

Код, написаний на C++, компілюється безпосередньо в машинний код перед виконанням. Це дозволяє програмам працювати швидше, оскільки інтерпретація під час виконання не потрібна. У мовах з інтерпретацією (як-от Python) програма працює повільніше, оскільки код виконується рядок за рядком під час роботи програми.

- **Гнучка оптимізація коду**

Компілятори C++ (наприклад, GCC, Clang) забезпечують безліч опцій для оптимізації коду під час компіляції. Це дозволяє отримати максимально ефективний машинний код, що може істотно підвищити швидкість виконання алгоритмів.

- **Можливість ручної оптимізації**

Мова C++ дозволяє програмістам проводити ручну оптимізацію, враховуючи деталі архітектури процесора і оперативної пам'яті. Це включає використання інструкцій SIMD (Single Instruction, Multiple Data), маніпуляції з кешем процесора та інші трюки для зменшення часу виконання.

Усе це робить C++ ідеальним вибором для алгоритмічних змагань, де важливо не тільки правильність, але й ефективність рішень. Саме тому програмісти, які прагнуть досягти успіху на олімпіадах з програмування, часто використовують C++ як основну мову.

Використані джерела інформації

1. Стеценко А. Програмування на С++ для початківців. Видавництво «Освіта України», 2010.
2. Наконечний О. Основи об’єктно-орієнтованого програмування на С++. Видавництво «Львівська Політехніка», 2012.
3. Висоцький В. Програмування мовою С++: теорія та практика. Видавництво «Київський університет», 2015.
4. Караванова Т. П. Програмування на С++ для студентів технічних спеціальностей. Видавництво «НТУУ КПІ», 2014.
5. Караванова Т. П. 777 задач з програмування на С++. Видавництво «НТУУ КПІ», 2015.
6. Караванова Т. П. Методи побудови алгоритмів і їх аналіз. Видавництво «Львівська політехніка», 2016.
7. Биков В. Ю. Основи програмування мовою С++. Видавництво «Інститут педагогіки», 2016.
8. Стівен Прата. Мова програмування С++ лекції та практика (6-е видання).
9. Kernighan B., & Ritchie D. The C Programming Language. Prentice Hall, 1988.
10. Stroustrup B. The C++ Programming Language. Addison-Wesley, 2013.
11. Smiley J. Learning programming on C++. Course Technology, 2002.
12. Deitel H. M. How to Program in C++. Pearson Education, 2016.

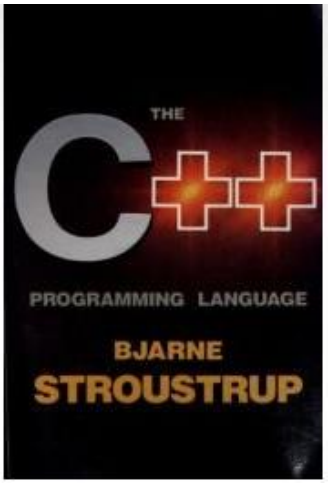
Додатки

Додаток 1. Рекомендована література до опрацювання

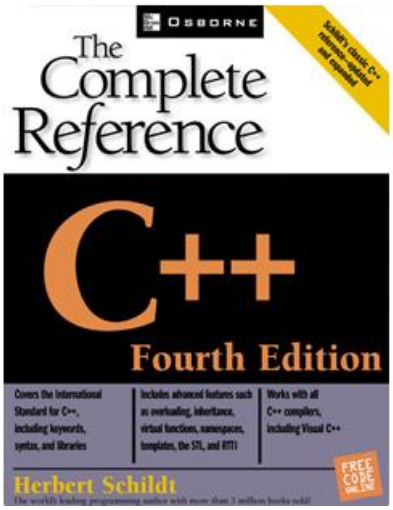
Якщо вас зацікавила концепція реалізація задач мовою програмування C++, то вам слід обов’язково поглиблювати свої знання та познайомитися з відповідною навчальною літературою.

Б'ярн Страуструп – "Програмування на C++"

Це класичний підручник від самого творця мови C++, який детально розглядає принципи програмування на цій мові. Книга охоплює основи та просунуті аспекти C++ і є ідеальним ресурсом для як новачків, так і досвідчених програмістів та обов’язково має бути в арсеналі програміста.



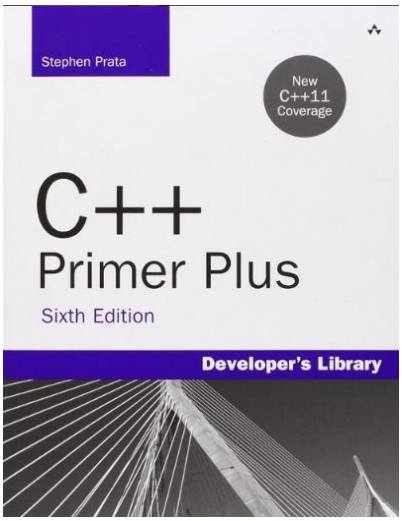
Герберт Шилдт – "C++: Повний курс"



Один із найвідоміших посібників для вивчення C++. Шилдт пояснює базові концепції мови, а також пропонує практичні приклади та поради для розв'язання реальних програмістських задач. Містить серйозне пояснення структур даних.

Стівен Прата – "Мова програмування C++"

Дуже популярний підручник, який підходить як для початківців, так і для тих, хто вже має певні знання в програмуванні. У книзі представлені як базові поняття, так і складніші теми, що допоможуть глибше зрозуміти роботу мови, якщо в цьому виникне необхідність



Скотт Мейерс – "Ефективне використання C++"

Ця книга містить цінні рекомендації щодо вдосконалення коду та підвищення ефективності програмування на C++. Вона допоможе уникати поширених помилок та використовувати найкращі практики програмування.

Андрій Стеценко – "Програмування на C++ для початківців"

Підручник, написаний українським автором, орієнтований на початківців. Він простою мовою пояснює основи C++ та супроводжується прикладами і завданнями для самостійного розв'язку.

Додаток 2. Бібліотеки

Механізм підключення бібліотек через #include дозволяє використовувати готові рішення, написані іншими розробниками та спрощує створення програм.

Підключення бібліотек у програмах на C++ працює через директиву препроцесора #include, яка вказує компілятору на те, що в програмі використовуються певні зовнішні файли або бібліотеки.

Під час написання олімпіад з програмування найчастіше доводиться використовувати наступні бібліотеки:

#include<iostream>: Підключає заголовковий файл iostream, який містить оголошення потоків введення та виведення:

- std::cin: Стандартний потік для введення через консоль.
- std::cout: Стандартний потік для виведення в консоль.
- std::endl: Маніпулятор, який додає символ нового рядка та очищає потік.

#include <algorithm>: є частиною стандартної бібліотеки і містить набір алгоритмів, які можна використовувати для роботи з колекціями даних (масивами, векторами, списками тощо). Ці алгоритми охоплюють широкий спектр операцій, таких як сортування, пошук, перестановки, обчислення і копіювання елементів.

#include <vector> - містить визначення шаблонного класу std::vector, який реалізує динамічний масив. Клас std::vector дозволяє зберігати елементи в динамічній послідовності з доступом за індексом, подібно до звичайного масиву, але з додатковими можливостями.

Наприклад:

```
1. #include <iostream>
2. #include <algorithm>
3. #include <vector>
4.
```

```
 5. int main() {
 6.     std::vector<int> vec = {3, 1, 4, 1, 5, 9};
 7.     std::sort(vec.begin(), vec.end());  // Сортування
вектора
 8.     for (int x : vec) {
 9.         std::cout << x << " ";  // Виведення відсортованих
елементів
10.     }
11.     return 0;
12. }
```

#include <cmath>: містить різноманітні математичні функції, константи та типи, які використовуються для виконання складних обчислень і надає програмістам можливість використовувати функції для роботи з числами, наприклад тригонометричні, експоненційні, логарифмічні та інші математичні операції.

До складу цієї бібліотеки включено досить потужний арсенал функцій. які стануть в нагоді при виконанні завдань:

- Арифметичні функції:

- std::abs(x) — повертає абсолютне значення числа (для цілих і дійсних чисел).
- std::pow(x, y) — підносить число x до степеня y.
- std::sqrt(x) — обчислює квадратний корінь з числа x.
- std::cbrt(x) — обчислює кубічний корінь з числа x.

- Тригонометричні функції:

- std::sin(x) — синус кута x (в радіанах).
- std::cos(x) — косинус кута x (в радіанах).
- std::tan(x) — тангенс кута x (в радіанах).

- Округлення та маніпуляції з плаваючими числами:

- std::ceil(x) — округляє число x в більший бік до найближчого цілого.
- std::floor(x) — округляє число x в менший бік до найближчого цілого.
- std::round(x) — округляє число x до найближчого цілого.
- std::trunc(x) — обрізає десяткову частину числа x, залишаючи тільки цілу.

Це лише деякі операції, а тому для більш серйозного використання бібліотека потребує глибокого аналізу.

#include <string>: використовується для роботи з символьними рядками і надає зручний спосіб маніпулювати текстовими даними.

Наведемо приклади деяких корисних методів:

- length() або size(): повертає кількість символів у рядку.

```
1. std::string str = "Hello";
2. std::cout << str.length();  // Виведе 5
```

- empty(): перевіряє, чи є рядок порожнім.
- append() : додає рядок або символ до кінця існуючого рядка.

```
1. std::string words = "Hello";
2. words.append(" World").append(1, '!');
```

У цьому прикладі рядок "Hello" спочатку доповнюється " World", а потім додається символ '!', результатом буде "Hello World!".

#include <cstdlib>: містить функції для роботи із загальними завданнями, пов'язаними з керуванням пам'яттю, генерацією випадкових чисел, роботою з середовищем виконання, а також перетворенням типів.

#include <bits/stdc++.h> — це нестандартна бібліотека, яка використовується в основному в змаганнях з програмування для спрощення підключення стандартних бібліотек C++. Включає майже всі заголовкові файли

стандартної бібліотеки C++, що дозволяє уникнути необхідності явно підключати кожну бібліотеку окремо.

Серед включених у цей заголовковий файл бібліотек є iostream, vector, algorithm, cmth багато інших, а тому потребує більш глибшого аналізу. Додатково слід розуміти, що ця бібліотека не входить до специфікації мови C++. Вона доступна лише в деяких компіляторах (наприклад, GCC), тому завжди слід тестувати перед початком роботи вашу машину на відповідну можливість з нею працювати.

Слід пам'ятати, що підключення такої бібліотеки додає до програми необхідність мати справу зі збільшеними об'ємами інформації, що у свою чергу не дає можливість економити час і ресурси в тому об'ємі, якого вимагають рішення олімпіадних завдань.

Примітка

C++ для стандартних бібліотек використовують заголовкові файли без розширення .h. Наприклад, <iostream>, <vector>, <algorithm>, C++ є наступником мови C, в якої у свою чергу деякі заголовкові файли зберегли своє розширення .h для сумісності. Прикладом є такі бібліотеки, як <stdio.h>, <stdlib.h>.

Слід пам'ятати що іноді, особливо в довідковій літературі може зустрічатись підключення бібліотек з C, в цьому випадку це матиме наступний вигляд:

- *#include <math.h> // Заголовок із C*
- *#include <cmath> // Заголовок для C++*

Відомості про авторів

Карявка Сергій Сергійович – старший учитель, методист, учитель інформатики вищої кваліфікаційної категорії ліцею "Інтелект" Долинської районної ради.

Паламарюк Максим Русланович – Senior Big Data Engineer, студент магістерської програми "Data Science" в Українському Католицькому Університеті та сертифікований спеціаліст з хмарних технологій на AWS та GCP, переможець Всеукраїнської олімпіади з ІТ 2019 в м. Дніпро

C++ в задачах. Олімпійські кроки

Підписано до друку 07.07.2025 р.
Формат 60х84 1/16. Папір офсетний.
Гарнітура «Times New Roman».
Друк – принтер. Тираж – 100 прим.
Зам. № 468

КЗ «КОІППО імені Василя Сухомлинського», вул. Велика Перспективна, 39/63, Кропивницький, 25006

Віддруковано в лабораторії інформаційно-методичного забезпечення освітнього процесу КЗ «КОІППО імені Василя Сухомлинського», вул. Велика Перспективна, 39/63, Кропивницький, 25006