Код програми реалізує систему обчислень із розподіленими компонентами, де кожен компонент виконує обчислення, а менеджер обчислень керує групами та їх компонентами.

В коді використовуються потоки для паралельного виконання обчислень компонентами. Для цього використовується стандартна бібліотека `<thread>` в C++. Це дозволяє одночасно виконувати декілька операцій на різних ядрах процесора, підвищуючи ефективність програми, особливо при виконанні ресурсоємних обчислень.

Для зберігання компонентів, що належать до певної групи, використовується контейнер `std::map`, який зберігає пари ключ-значення, де ключем є індекс групи, а значенням — вектор компонентів цієї групи.

Теоретичні аспекти використання std::map:

- **Асортимент**: `std::map` автоматично підтримує сортування за ключами, що робить його зручним для зберігання елементів з унікальними ідентифікаторами (у вашому випадку, індексами груп).
- **Час доступу**: Операції вставки, видалення і пошуку в `std::map` здійснюються за час O(log n), де n — кількість елементів у мапі. Це дозволяє ефективно управляти великими наборами даних.

Код містить нескладну командну систему, яка дозволяє створювати групи, додавати компоненти до груп, запускати обчислення, виводити підсумки тощо. Команди вводяться через консоль, що є простим інтерфейсом для користувача.
Теоретичні аспекти командного вводу:

- **Парсинг команд**: Строки команд парсяться за допомогою функцій, як-от `substr()` і `stoi()`, що дозволяє виділяти частини введеного тексту та перетворювати їх у відповідні значення.

Система обчислень та паралельний доступ до результатів

- Кожен компонент обчислює результат для конкретного значення `x`, що пов’язано з групою, і виводить його на екран після завершення. Це дозволяє моделювати паралельну обробку незалежних завдань.
- Після того як всі потоки завершили своє виконання, викликається метод `printSummary()`, який виводить підсумки обчислень.

Теоретичні аспекти обчислень та синхронізації:

- **Часова синхронізація** між потоками вимагає використання механізмів синхронізації, якщо потоки працюють з спільними ресурсами (наприклад, для запису результатів в загальний вектор). Однак у цьому коді кожен потік працює із своїм власним компонентом, і це зменшує потребу в синхронізації.

- **Завершення потоків**: Використання `join()` на потоці гарантує, що головний потік чекає на завершення всіх обчислень перед виведенням підсумків.

<u>Про реалізовану систему:</u>

1. Обчислювальні компоненти та їх організація
У коді створено клас `ComputationComponent`, який представляє обчислювальний компонент. Кожен компонент має символ (наприклад, `'A'`, `'B'`) і зберігає результат обчислення. В класі є метод `compute`, який моделює обчислення. У цьому випадку, метод `compute` обчислює квадрат числа, переданого як параметр `x`, і виводить результат.
- **Принцип роботи**: Клас `ComputationComponent` має механізм для моделювання обчислень, які тривають одну секунду (`std::this_thread::sleep_for(std::chrono::seconds(1))`), після чого компонент виводить результат у консоль.

2. Менеджер обчислень
Клас `ComputationManager` є відповідальним за керування обчисленнями та групами компонентів. Він має наступні функції:
- **Створення групи**: Використовуючи метод `createGroup`, можна створити групу, яка містить певне значення `x`. Групи використовуються для групування компонентів, щоб вони виконували обчислення з одним параметром.
- **Додавання компонентів**: За допомогою методу `addComponentToGroup` можна додавати компоненти до групи. Кожен компонент має символ і належить до конкретної групи.
- **Запуск обчислень**: Метод `run` запускає обчислення для всіх компонентів у групах. Це досягається за допомогою багатопоточності, де кожен компонент виконується в окремому потоці.
- **Виведення підсумків**: Метод `printSummary` виводить результати обчислень для всіх компонентів.

3. Багатопоточність
Використання багатопоточності є важливим аспектом коду. Метод `run` використовує об'єкти `std::thread`для паралельного виконання обчислень для кожного компонента.
- **Як це працює**: Для кожного компонента з групи створюється окремий потік, в якому виконується метод `compute`. Це дозволяє компонентам працювати паралельно, що може значно скоротити час виконання обчислень у порівнянні з послідовним виконанням кожного компонента.
- **Задача потоків**: Кожен потік виконує обчислення незалежно від інших, що дозволяє ефективно обробляти декілька груп компонентів одночасно. Після того, як всі потоки завершаться (це забезпечується викликом `join()` для кожного потоку), програма виводить повідомлення про завершення обчислень.

4. Командний інтерфейс

Програма реалізує простий командний інтерфейс для взаємодії з користувачем через консоль. Користувач може вводити різні команди:

- `group <x>` — створення групи з параметром `x`.
- `new <symbol> <group_idx>` — додавання нового компонента до групи з індексом `group_idx`.
- `run` — запуск всіх обчислень для всіх груп.
- `summary` — виведення підсумків для всіх компонентів.
- `exit` — завершення програми.

Це дозволяє користувачу динамічно створювати групи та компоненти і контролювати процес обчислень.

5. Масиви та мапи для зберігання даних

У програмі використовуються:

- **`std::vector<int> groups`**: цей масив зберігає значення `x` для кожної групи.
- **`std::map<int, std::vector<ComputationComponent*>> groupComponents`**: ця мапа відображає індекс групи на список вказівників на компоненти, що належать до цієї групи.
- **`std::vector<ComputationComponent*> finishedComponents`**: цей масив зберігає вказівники на компоненти, які завершили обчислення.

Завдяки використанню цих контейнерів програма може зберігати і організовувати компоненти, а також їх групи.