Київський національний університет імені Тараса Шевченка
Механіко-математичний факультет

Реферат на тему:

«Бітові поля»

Виконав:
студент 2 курсу, 1-ша групи комп’ютерної математики
Решетник Віталій

Київ 2022

ЗМІСТ

Вступ

Опис роботи бітових полей на Сі.

Опис поботи псевдоконтейнерів bitset.

Опис роботи vector<bool> на Сі++.

Порівняння роботи.

Висновки

Вступ

а)Під час виконання цієї роботи для кожного завдання потрібно написати програму на мові Сі в якій виконано розробку структури з бітовими полями відповідно до постановкою завдання, введення даних елементів структури, їх обробка і виведення на екран в зазначеному форматі. При цьому бітові поля можуть використовуватися тільки в якості полів структури. Введення даних здійснюється з клавіатури з урахуванням вимог до вхідних даних, що містяться в постановці завдання. Обмеженнями на вхідні дані є максимальна величина рядкових даних, діапазони числових типів полів структури, максимально дозволений розмір масиву структур в мові Сі

б) Замість бітової структури використовуйте колекцію vector<bool> зі стандартної бібліотеки

в) Замість бітової структури використовуйте псевдоконтейнер bitset зі стандартної бібліотеки

Опис роботи бітових полей на Сі.

Бітові поля забезпечують зручний доступ до окремих біт даних. Вони дозволяють формувати об'єкти з довжиною, не кратною байту. Бітові поля повинні оголошуватися як int, unsigned чи signed. Бітові поля довжиною 1 повинні бути оголошені як unsigned, оскільки 1 біт не може мати знака. Бітові поля можуть мати довжину від 1 до 16 біт для 16-бітових середовищ і від 1 до 32 біт для 32-бітових середовищ. Що дозволяє економити пам'ять, щільніше розміщуючи дані. Зазвичай це використовують у таких випадках:

1. Якщо обмежене місце для зберігання інформації, можна зберегти кілька логічних (істина/хиба) змінних в одному байті.
2. Деякі інтерфейси пристроїв передають інформацію, закодувавши біти за один байт.
3. Деяким процедурам кодування необхідно отримати доступ до окремих біт у байті.

Бітові поля мають деякі обмеження:

1. Не можна отримати адресу змінної бітового поля.
2. Змінні бітового поля не можуть бути поміщені в масив.
3. Переходячи з комп'ютера на комп'ютер не можна бути впевненим у порядку зміни бітів (зліва направо або праворуч наліво). Будь-який код, який використовує бітові поля, залежить від комп'ютера.
4. Бітове поле не може існувати саме по собі. Воно може бути лише елементом структури чи об'єднання.

Опис роботи псевдоконтейнера bitset.

У сучасних архітекторів комп'ютера найменша адресна одиниця пам'яті - байт. Оскільки всі об'єкти повинні мати унікальні адреса пам'яті, це означає, що об'єкти повинні бути щонайменше одним байтом. Для більшості типів змінних це нормально. Однак для логічних значень це трохи нераціонально. Логічні типи мають лише два стани: істина (1) або хиба (0). Для цього набору умов потрібен лише один біт. Однак якщо змінна повинна бути щонайменше байт, а байт - 8 біт, це означає, що логічне значення використовує 1 біт, а решта 7 залишається невикористаною.

У більшості випадків це нормально - зазвичай ми не настільки обмежені пам’яттю, щоб піклуватися про 7 загублених біт. Однак у деяких випадках з інтенсивним зберіганням для підвищення ефективності може бути корисно «упакувати» 8 окремих логічних значень в одному байті.

Це вимагає маніпулювання об'єктами на бітовому рівні. На щастя, C ++ дає інструмент для цього bitset.Основні команди для роботи з бітами в цьому класі:

test() дозволяє дізнатися, чи дорівнює біт 0 або 1;

set() дозволяє встановити біт 1 (вона нічого не зробить, якщо біт вже дорівнює 1);

reset() дозволяє скинути біт 0 (вона нічого не дасть, якщо біт вже дорівнює 0);

flip() дозволяє інвертувати значення біта з 0 на 1 або навпаки.

На відміну від бітових полів, кількість біт може бути будь-якою, які адресується за індексом.

Опис роботи vector<bool> на Ci++.

Для векторів, що містять елементи логічного типу, у стандартній бібліотеці C++ визначено спеціалізований різновид класу vector. Це було зроблено для того, щоб оптимізована версія займала менше місця, ніж стандартна реалізація vector типу bool. У стандартній реалізації для кожного елемента резервується щонайменше 1 байт, тут 1 біт. Але ця, спеціалізована, версія вектору вимагає спеціальної обробки посилань та ітераторів. В результаті vector<bool> не відповідає всім вимогам інших векторів зокрема: значення vector<bool>preference не є повноцінним l-байтним значенням, а ітератор vector<bool>::iterator не є ітератором довільного доступу. Клас vector може поступатися звичайним реалізаціям за швидкістю роботи, тому що операції з елементами доводиться перетворювати в операції з бітами. Втім, внутрішній пристрій залежить від реалізації, тому ефективність, як швидкість роботи, так і витрати пам'яті, можуть бути різними. Розмір vector<bool> змінюється динамічно, тому його можна розглядати як бітове поле з динамічним розміром. Інакше кажучи, можна додавати та видаляти біти.

Клас vector<bool> є чимось більшим, ніж спеціалізація класу vector<> для bool. До нього також включено підтримку спеціальних операцій, що спрощують роботу з наборами бітів:

a.flip()	Інвертує всі логічні елементи
a[idx].flip()	Інвертує логічний елемент із заданим індексом
a[idx] = val	Надає значення логічному елементу з індексом idx
a[idxl] = a[idx2]	Надає значення елемента з індексом idx2 елементу з індексом idxl

Порівняння роботи.

N1 (NZ1)

У цьому завданні бітові значення використовуються за прямим призначенням, присутність чи відсутність якоїсь ознаки об’єктів, що цілком реалізуємо за допомогою усіх трьох запропонованих варіантів виконання.

N2&3 (NZ2)

У цих завданнях іде робота зі змінними цілочисленого формату. Доцільне використання бітових полів, так як вони надають доступ до необхідних бітів числа через об’єднання тому й можна легко перевірити чи кратне число 8, чи парне воно, просто перевіряючі певні біти на ноль. Це не є реалізуємим з bitset та vector, тому що останні не можуть бути членом об’єднання.

N4 (NZ4)

bitset та vector не дають змоги створити змінні цілого типу необхідної бітової довжини наприклад: для хвилин потрібно 6 біт 2^6=64>60, коли для годин 5 біт 2^5=32>24. Для цього доцільно використовувати бітові поля, так як вони дають можливість створити змінну цілого типу необхідного розміру.

Висновки

Підсумовуючи. Бітові поля зручно використовувати як змінну цілогочисленого типу заданого розміру, але вони обмежені розрядністю і не можуть бути використанні не в складі структури чи об’єднання. Bitset можна адресувати як масив та може бути будь-якого сталого розміру. vector<bool> дає можливість динамічно змінювати розмір.