202505262333

CVector: Ваш удобный массив (но лучше!)

Что это за проект?

Представьте, что у вас есть коробка (массив) для хранения вещей. Этот проект предоставляет вам удобную и простую в использовании коробку под названием CVector. Это как обычный массив в C++, но он умнее и безопаснее. Он автоматически управляет пространством внутри коробки, так что вам не нужно беспокоиться о нехватке места или случайной потере ваших вещей.

Почему использовать CVector вместо обычного массива?

- Почти нет ограничений по размеру: Обычные массивы требуют фиксированного размера при создании. CVector может увеличиваться или уменьшаться по мере необходимости.
- Безопасность прежде всего: CVector помогает предотвратить распространенные ошибки, которые возникают при работе с массивами напрямую, такие как попытка поместить что-то за пределами коробки (ошибки выхода за границы) или забыть очистить после завершения (утечки памяти).
- Простота использования: CVector имеет встроенные инструменты для сортировки ваших элементов, поиска конкретных элементов и даже выполнения математических операций с другими CVectors.

Что вам нужно знать перед началом

- Основы C++: Вы должны знать основы C++, такие как переменные, типы данных (int, float, string и т.д.), и как писать простые функции.
- Шаблоны: CVector может содержать элементы любого типа. Это реализовано с помощью так называемых "шаблонов". Представьте это как форму, которая может создавать коробки для разных вещей. Вы увидите , , и т.д. в коде. Это говорит CVector, какие виды вещей он будет содержать.

Как использовать CVector

1. Включите заголовочный файл:

В начале вашего файла .cpp добавьте эту строку:
#include "CVector.h"

Это сообщает вашей программе, где найти определение класса CVector.

2. Создайте CVector:

Есть несколько способов создать CVector:

- Пустой CVector: Создает CVector без элементов.

```
CVector<int> myNumbers; // CVector для хранения целых чисел
CVector<string> myWords; // CVector для хранения строк
```

- CVector определенного размера: Создает CVector, который может сразу содержать определенное количество элементов. Все места автоматически устанавливаются в значение по умолчанию (обычно 0 для чисел, пустые строки для строк).

```
CVector<int> scores(10); // CVector, который может содержать 10 целых чисел
```

- CVector из обычного массива: Копирует элементы из обычного массива C++ в новый CVector.

```
int numbers[] = {1, 2, 3, 4, 5};
CVector<int> myNumbers(numbers, 5); // Создает CVector с числами от 1 до 5
```

3. Добавьте или измените элементы в CVector:

Используйте [] (квадратные скобки), чтобы получить доступ к элементам в CVector, как в обычном массиве. Помните, что первый элемент находится на позиции 0, второй на позиции 1 и так далее.

```
CVector<int> ages(3); // Создаем CVector для 3 возрастов
ages[0] = 25; // Устанавливаем первый возраст на 25
ages[1] = 30; // Устанавливаем второй возраст на 30
ages[2] = 40; // Устанавливаем третий возраст на 40
cout << ages[1]; // Выводит 30
```

Важно: Если вы попытаетесь получить доступ к элементу за пределами допустимого диапазона (0 до size - 1), программа выдаст ошибку и потенциально завершится аварийно.

4. Получите размер CVector:

Используйте метод getsize(), чтобы узнать, сколько элементов может содержать CVector в данный момент.

```
CVector<int> values(5);
int count = values.getsize(); // count будет 5
cout << "CVector может содержать " << count << " значений." << endl;
```

5. Измените размер CVector:

Используйте метод setsize(), чтобы изменить количество элементов, которые может содержать CVector.

```
CVector<int> grades(3);
grades.setsize(5); // Теперь CVector может содержать 5 оценок
```

Важно:

- Если вы увеличите CVector, новые места будут заполнены значением по умолчанию (0 для чисел, пустые строки для строк и т.д.)
- Если вы уменьшите CVector, элементы в конце будут удалены и потеряны.

6. Заполните CVector случайными значениями:

Используйте метод random(min, max), чтобы заполнить массив случайными значениями.

```
CVector<int> randomNumbers(10);
randomNumbers.random(1, 100); // Заполняет 10 случайными числами от 1 до 100
```

7. Сортировка CVector:

- Сортировка по возрастанию: Сортирует элементы от наименьшего к наибольшему.

```
CVector<int> numbers = {5, 2, 8, 1, 9};
numbers.SortUp(); // numbers теперь будет {1, 2, 5, 8, 9}
```

- Сортировка по убыванию: Сортирует элементы от наибольшего к наименьшему.

```
CVector<int> numbers = {5, 2, 8, 1, 9};
```

```
numbers.SortDown(); // numbers теперь будет {9, 8, 5, 2, 1}
```

8. Найдите элемент в CVector:

Используйте метод FindElement(value), чтобы искать конкретный элемент.

```
CVector<int> scores = {85, 92, 78, 95, 88};
int* location = scores.FindElement(92);
if (location != nullptr) {
    cout << "Найдено 92 на позиции: " << (location - &scores[0]) << endl; //
Показать индекс найденного элемента
} else {
    cout << "92 не найдено в CVector." << endl;
}
```

Важно:

- FindElement() возвращает указатель на элемент, если он найден. Указатель — это как адрес в памяти.
- Если элемент не найден, FindElement() возвращает nullptr (означает "ничего"). Всегда проверяйте, является ли результат nullptr, прежде чем использовать указатель!
- (location - &scores[0]) вычисляет индекс.

9. Выполните математические операции с CVector:

Вы можете складывать или вычитать два CVector вместе, если они имеют одинаковый размер. Результат — это новый CVector, где каждый элемент является суммой или разностью соответствующих элементов в исходных CVector.

```
CVector<int> a = {1, 2, 3};
CVector<int> b = {4, 5, 6};
CVector<int> sum = a + b; // sum будет {5, 7, 9}
CVector<int> difference = b - a; // difference будет {3, 3, 3}
```

Важно: Если вы попытаетесь сложить или вычесть CVector разного размера, программа выдаст ошибку.

10. Вывод CVector в консоль

Вы можете распечатать содержимое CVector, используя оператор <<.

```
CVector<int> myVector = {1, 2, 3, 4, 5};
cout << myVector; // Выводит элементы CVector в консоль
```

11. Ввод CVector из консоли

Вы можете заполнить CVector значениями от пользователя, используя оператор >>.

```
CVector<int> myVector;
cin >> myVector; // Предлагает пользователю ввести размер и элементы CVector
cout << myVector; // Выводит введенные значения
```

Обработка ошибок:

CVector использует "исключения", чтобы сообщить вам, когда что-то идет не так. Вам нужно поместить ваш код CVector внутрь блока try...catch, чтобы обрабатывать эти ошибки корректно.

```
try {
    CVector<int> values(5);
    values[10] = 100; // Это вызовет ошибку!
} catch (const std::exception& error) {
    cout << "Произошла ошибка: " << error.what() << endl;
}
catch (const std::out_of_range& error) {
    std::cout << std::endl << "Ошибка! " << error.what();
}
```

Распространенные ошибки:

- std::out_of_range: Вы попытались получить доступ к элементу за пределами допустимого диапазона (0 до size - 1).
- std::exception: Произошла общая ошибка. Сообщение в error.what() даст вам больше деталей.

Пример кода:

```
#include <iostream>
#include "CVector.h"
int main() {
    try {
        // Создаем CVector для хранения 5 целых чисел
        CVector<int> myNumbers(5);
```

```
        // Заполняем его случайными числами от 1 до 100
        myNumbers.random(1, 100);
        // Печатаем числа
        std::cout << "Мои числа: " << myNumbers << std::endl;
        // Сортируем числа от наименьшего к наибольшему
        myNumbers.SortUp();
        std::cout << "Отсортированные числа: " << myNumbers << std::endl;
        // Пытаемся получить доступ к элементу за пределами допустимого диапазона
        std::cout << "Третий элемент: " << myNumbers[2] << std::endl; // это
нормально
    // std::cout << "Десятый элемент: " << myNumbers[10] << std::endl; // это
вызовет исключение!
    } catch (const std::exception& error) {
        std::cerr << "Ошибка: " << error.what() << std::endl;
    }
    catch (const std::out_of_range& error) {
        std::cerr << "Ошибка: " << error.what() << std::endl;
    }
    return 0;
}
```

Где найти код:

- CVector.h: Этот файл содержит определение класса CVector (что он может делать).
- main.cpp: Этот файл содержит пример использования класса CVector.