CPP98: std::vector

📦 Header

```
#include <vector>
```

⚙️ Declaration

```
std::vector<Type> vec;
```

🛠️ Common Methods

- `push_back(val)` - Add element to end
- `pop_back()` - Remove last element
- `size()` - Number of elements
- `empty()` - Check if empty
- `clear()` - Remove all elements
- `at(index)` - Access with bounds checking (throws exception)
- `operator[]` - Access without bounds checking
- `front()` - First element
- `back()` - Last element
- `begin(), end()` - Iterators to start/end
- `rbegin(), rend()` - Reverse iterators

🔔 Basic Usage

```
std::vector<int> numbers;
numbers.push_back(42);
numbers.push_back(21);
numbers.push_back(10);

// Access elements
for (size_t i = 0; i < numbers.size(); i++)
    std::cout << numbers[i] << std::endl;
```

🔔 Iterator Usage

```
std::vector<int>::iterator it;
for (it = numbers.begin(); it != numbers.end(); ++it)
    std::cout << *it << std::endl;

// Const iterator
std::vector<int>::const_iterator cit;
for (cit = numbers.begin(); cit != numbers.end(); ++cit)
    std::cout << *cit << std::endl;
```

🔔 Initialization

```
std::vector<int> vec;                    // Empty vector
std::vector<int> vec(10);                // 10 elements, default initial
std::vector<int> vec(10, 42);            // 10 elements, all = 42
std::vector<int> vec2(vec);              // Copy constructor
std::vector<int> vec3(vec.begin(), vec.end());  // Range constructor
```

Erase Example

```
std::vector<int>::iterator it;
for (it = vec.begin(); it != vec.end(); ) {
    if (*it == 42)
        it = vec.erase(it);  // erase returns next valid iterator
    else
        ++it;
}
```

42 Webserv Example

```
class Server {
private:
    std::vector<Client> clients;

public:
    void addClient(const Client& client) {
        clients.push_back(client);
    }

    void removeClient(int fd) {
        std::vector<Client>::iterator it;
        for (it = clients.begin(); it != clients.end(); ++it) {
            if (it->getFd() == fd) {
                clients.erase(it);
                break;  // Important: iterator invalidated after erase
            }
        }
    }

    Client* findClient(int fd) {
        for (size_t i = 0; i < clients.size(); i++) {
            if (clients[i].getFd() == fd)
                return &clients[i];
        }
        return NULL;
    }
};
```

Memory Management

```
vec.capacity()      // Allocated storage
vec.reserve(n)      // Pre-allocate memory for n elements
vec.resize(n)       // Change size (adds/removes elements)
```

```
vec.resize(n, val)  // Change size, new elements = val

// Example: Avoid reallocations
std::vector<int> vec;
vec.reserve(1000);  // Allocate space for 1000 elements upfront
for (int i = 0; i < 1000; i++)
    vec.push_back(i);  // No reallocation needed
```

⚠ Gotchas

- Iterators invalidate after `push_back/erase/resize`
- `operator[]` doesn't check bounds (use `at()` for safety)
- `push_back` can reallocate entire vector (slow if frequent)
- Use `reserve()` if you know approximate size
- Vector of pointers: you must delete manually

 CPP98 Constraints

- No range-based for loops (C++11)
- No auto keyword
- No initializer lists `{1, 2, 3}`
- No `emplace_back` (use `push_back`)

 Related

map, set, list, deque, iterator, algorithm