

Model:TheB.AI Free

Problem Statement: Implement Deque using C++. C++ STL is NOT allowed (vectors also not allowed, design your own if required)

Operations : The C++ standard specifies that a legal (i.e., standard-conforming) implementation of deque must satisfy the following performance requirements (consider the data type as T) :

1. deque() - initialize an empty deque. Time complexity: O(1)
2. deque(n) - initialize a deque of length n with all values as default value of T. Time complexity: O(n)
3. deque(n, x) - Initialize a deque of length n with all values as x. Time complexity: O(n)
4. bool push_back(x) - append data x at the end. Return true if operation is performed successfully, else return false. Time complexity: O(1)
5. bool pop_back() - erase data at the end. Return true if operation is performed successfully, else return false. Time complexity: O(1)
6. bool push_front(x) - append data x at the beginning. Return true if operation is performed successfully, else return false. Time complexity: O(1)
7. bool pop_front() - erase an element from the beginning. Return true if operation is performed successfully, else return false. Time complexity: O(1)
8. T front() - returns the first element(value) in the deque. If the first element is not present, return the default value of T. Time complexity: O(1)
9. T back() - returns the last element(value) in the deque. If the last element is not present, return the default value. Time complexity: O(1)
10. T D[n] - returns the nth element of the deque. You need to overload the [] operator. If nth element is not present return default value of T. Time complexity: O(1)
11. bool empty() - returns true if deque is empty else returns false. Time complexity: O(1)
12. int size() - returns the current size of deque. Time complexity: O(1)
13. void resize(n) - change the size dynamically to new size n. Time complexity: O(n)
 - If the new size n is greater than the current size of the deque, then insert new elements with the default value of T at the end of the queue.
 - If the new size n is smaller than the current size, then keep n elements from the beginning of the deque.
14. void resize(n, d) - change the size dynamically to new size n. Time complexity: O(n)
 - If the new size n is greater than the current size of the deque, then insert new elements with value d at the end of the queue.
 - If the new size n is smaller than the current size, then keep n elements from the beginning of the deque.
15. void reserve(n) : change the capacity of deque to n, if n > current capacity; otherwise do nothing. Time complexity: O(n)
16. void shrink_to_fit() - reduce the capacity of the deque to current size. Time Complexity: O(size())
17. void clear() - remove all elements of deque. Time complexity: O(1)
18. int capacity() - return the current capacity of deque. Time complexity: O(1)

Input Format:

Design an infinitely running menu-driven main function. Each time the user inputs an integer corresponding to the serial number of the operation listed above. Then, take necessary arguments related to the selected operation and execute the respective method. Finally, the program must exit with status code 0, when 0 is provided as a choice.

Evaluation parameters:

Accuracy of operations and performance.

Note :

1. Your deque should be generic type i.e. it should be datatype independent and can support primitive data types like integer, float, string, etc. Hint: Use template in C++
2. For 1, 2 & 3 You can either define a constructor for the class or initialize the class object using void return type functions.
3. C++ STL is NOT allowed (including vectors, design your own if required)
4. D[0] - element at index 0 (i.e. first element from the front), D[1] - element at index 1 (i.e. second element from the front), D[-1] - element at last index (i.e. first element from the back), D[-2] - element at second last index (i.e. second element from the back)
5. Size of the deque is the number of elements currently present in your deque.
6. Capacity of the deque is the number of elements your deque can accommodate with currently held memory.
7. During Operation 1 both size and capacity of the deque should be set to zero.
8. If size is equal to capacity and a new element is inserted, then the capacity is doubled, unless capacity is zero, then it will become one.
9. If you have doubts about deciding the new capacity in any of the operations, refer to the behavior of the member functions of STL vector containers.

2023-08-13 15:10:16

Copy



To implement a deque without using the C++ STL, we can create a custom class called Deque. The Deque class should contain a dynamic array to store the elements of the deque, as well as variables to keep track of the size and capacity of the deque. Additionally, the Deque class should have a template parameter to support generic data types.

Here is a possible implementation of the Deque class:

cppCopy code

```
template <class T>
class Deque {
private:
    T* arr;
    int size;
    int capacity;

public:
    // Constructor
    Deque() {
        arr = nullptr;
        size = 0;
        capacity = 0;
    }

    // Destructor
    ~Deque() {
        delete[] arr;
    }

    // Initialize an empty deque
    void initialize() {
        size = 0;
        capacity = 0;
        delete[] arr;
        arr = nullptr;
    }

    // Initialize a deque of length n with default values of T
    void initialize(int n) {
        size = n;
        capacity = n;
        delete[] arr;
        arr = new T[n];
        for (int i = 0; i < n; i++) {
            arr[i] = T();
        }
    }

    // Initialize a deque of length n with all values as x
    void initialize(int n, T x) {
        size = n;
        capacity = n;
        delete[] arr;
        arr = new T[n];
        for (int i = 0; i < n; i++) {
            arr[i] = x;
        }
    }

    // Append data x at the end
    bool push_back(T x) {
        if (size == capacity) {
            int newCapacity = (capacity == 0) ? 1 : 2 * capacity;
            reserve(newCapacity);
        }
        arr[size] = x;
        size++;
        return true;
    }
}
```

```
// Erase data at the end
bool pop_back() {
    if (size > 0) {
        size--;
        return true;
    }
    return false;
}

// Append data x at the beginning
bool push_front(T x) {
    if (size == capacity) {
        int newCapacity = (capacity == 0) ? 1 : 2 * capacity;
        reserve(newCapacity);
    }
    for (int i = size; i > 0; i--) {
        arr[i] = arr[i - 1];
    }
    arr[0] = x;
    size++;
    return true;
}

// Erase an element from the beginning
bool pop_front() {
    if (size > 0) {
        for (int i = 1; i < size; i++) {
            arr[i - 1] = arr[i];
        }
        size--;
        return true;
    }
    return false;
}

// Returns the first element in the deque
T front() {
    if (size > 0) {
        return arr[0];
    }
    return T();
}

// Returns the last element in the deque
T back() {
    if (size > 0) {
        return arr[size - 1];
    }
    return T();
}

// Returns the nth element of the deque
T operator[](int n) {
    if (n >= 0 && n < size) {
        return arr[n];
    }
    return T();
}

// Returns true if deque is empty, else returns false
bool empty() {
    return size == 0;
}

// Returns the current size of deque
int getSize() {
    return size;
}
```

```
// Change the size dynamically to new size n
void resize(int n) {
    if (n > size) {
        if (n > capacity) {
            reserve(n);
        }
        for (int i = size; i < n; i++) {
            arr[i] = T();
        }
    }
    size = n;
}

// Change the size dynamically to new size n and insert value d
void resize(int n, T d) {
    if (n > size) {
        if (n > capacity) {
            reserve(n);
        }
        for (int i = size; i < n; i++) {
            arr[i] = d;
        }
    }
    size = n;
}

// Change the capacity of deque to n, if n > current capacity
void reserve(int n) {
    if (n > capacity) {
        T* newArr = new T[n];
        for (int i = 0; i < size; i++) {
            newArr[i] = arr[i];
        }
        delete[] arr;
        arr = newArr;
        capacity = n;
    }
}

// Reduce the capacity of the deque to current size
void shrink_to_fit() {
    if (size < capacity) {
        T* newArr = new T[size];
        for (int i = 0; i < size; i++) {
            newArr[i] = arr[i];
        }
        delete[] arr;
        arr = newArr;
        capacity = size;
    }
}

// Remove all elements of deque
void clear() {
    size = 0;
}

// Return the current capacity of deque
int getCapacity() {
    return capacity;
}

};
```

In the main function, you can create an instance of the Deque class and implement the menu-driven logic to call the respective methods based on user input. The main function should run indefinitely until the user inputs 0 to exit the program.



"In the main function, you can create an instance of the Deque class and implement the menu-driven logic to call the respective methods based on user input. The main function should run indefinitely until the user inputs 0 to exit the program.", is your response. How do i do it?

2023-08-13 15:16:52 Copy



To implement the menu-driven logic in the main function, you can follow these steps:

1. Import the necessary libraries or modules. For example, if you are using the deque data structure from the collections module, you can import it with `from collections import deque`.
2. Define the main function where you will implement the menu-driven logic.
3. Inside the main function, create an instance of the Deque class using `my_deque = deque()`.
4. Use a while loop to run the main logic indefinitely until the user inputs 0 to exit the program. For example:

python Copy code

```
while True:
    # Display the menu options
    print("Menu:")
    print("1. Add element to the deque")
    print("2. Remove element from the deque")
    print("3. View the deque")
    print("0. Exit the program")

    # Get user input for the menu choice
    choice = int(input("Enter your choice: "))

    # Implement the menu-driven logic
    if choice == 1:
        item = input("Enter the element to add: ")
        my_deque.append(item)
        print("Element added to the deque.")

    elif choice == 2:
        if len(my_deque) == 0:
            print("The deque is empty.")
        else:
            item = my_deque.popleft() # Or use pop() if you want to remove from the right end
            print(f"The removed element is: {item}")

    elif choice == 3:
        print(f"The current deque: {list(my_deque)}")

    elif choice == 0:
        print("Exiting the program...")
        break # Exit the while loop and terminate the program

    else:
        print("Invalid choice. Please try again.")
```

Note: In the above example, I assume you are using the deque from the collections module. If you are using a custom Deque class, adjust the code accordingly.

With this implementation, the program will display the menu, take user input for the desired operation, perform the corresponding action based on the user's choice, and continue to loop until the user enters 0 to exit the program.



2023-08-13 15:18:24 Copy Plaintext

"In the main function, you can create an instance of the Deque class and implement the menu-driven logic to call the respective methods based on user input. The main function should run indefinitely until the user inputs 0 to exit the program.", is your response. How do i do it? FOR THE DEQUE TEMPLATE ONLY GIVEN BY YOU AS 1st Response

2023-08-13 15:19:24 Copy



Here is an example of how you can implement the menu-driven logic for the Deque class in the main function:

cpp Copy code

```
#include <iostream>
#include "Deque.h"

int main() {
    Deque<int> deque;
    int choice, element;

    while (true) {
        std::cout << "Menu:\n";
        std::cout << "1. Insert element at front\n";
        std::cout << "2. Insert element at back\n";
        std::cout << "3. Remove element from front\n";
        std::cout << "4. Remove element from back\n";
        std::cout << "5. Display deque\n";
        std::cout << "0. Exit\n";
        std::cout << "Enter your choice: ";
        std::cin >> choice;

        switch (choice) {
            case 1:
                std::cout << "Enter element to insert: ";
                std::cin >> element;
                deque.insertFront(element);
                break;
            case 2:
                std::cout << "Enter element to insert: ";
                std::cin >> element;
                deque.insertBack(element);
                break;
            case 3:
                if (!deque.isEmpty())
                    deque.removeFront();
                else
                    std::cout << "Deque is empty.\n";
                break;
            case 4:
                if (!deque.isEmpty())
                    deque.removeBack();
                else
                    std::cout << "Deque is empty.\n";
                break;
            case 5:
                deque.display();
                break;
            case 0:
                std::cout << "Exiting program.\n";
                return 0;
            default:
                std::cout << "Invalid choice.\n";
        }
    }
}
```

This implementation will display a menu to the user and prompt for their choice. It then performs the corresponding operation on the deque based on their input. The program will continue to run until the user inputs 0 to exit.

