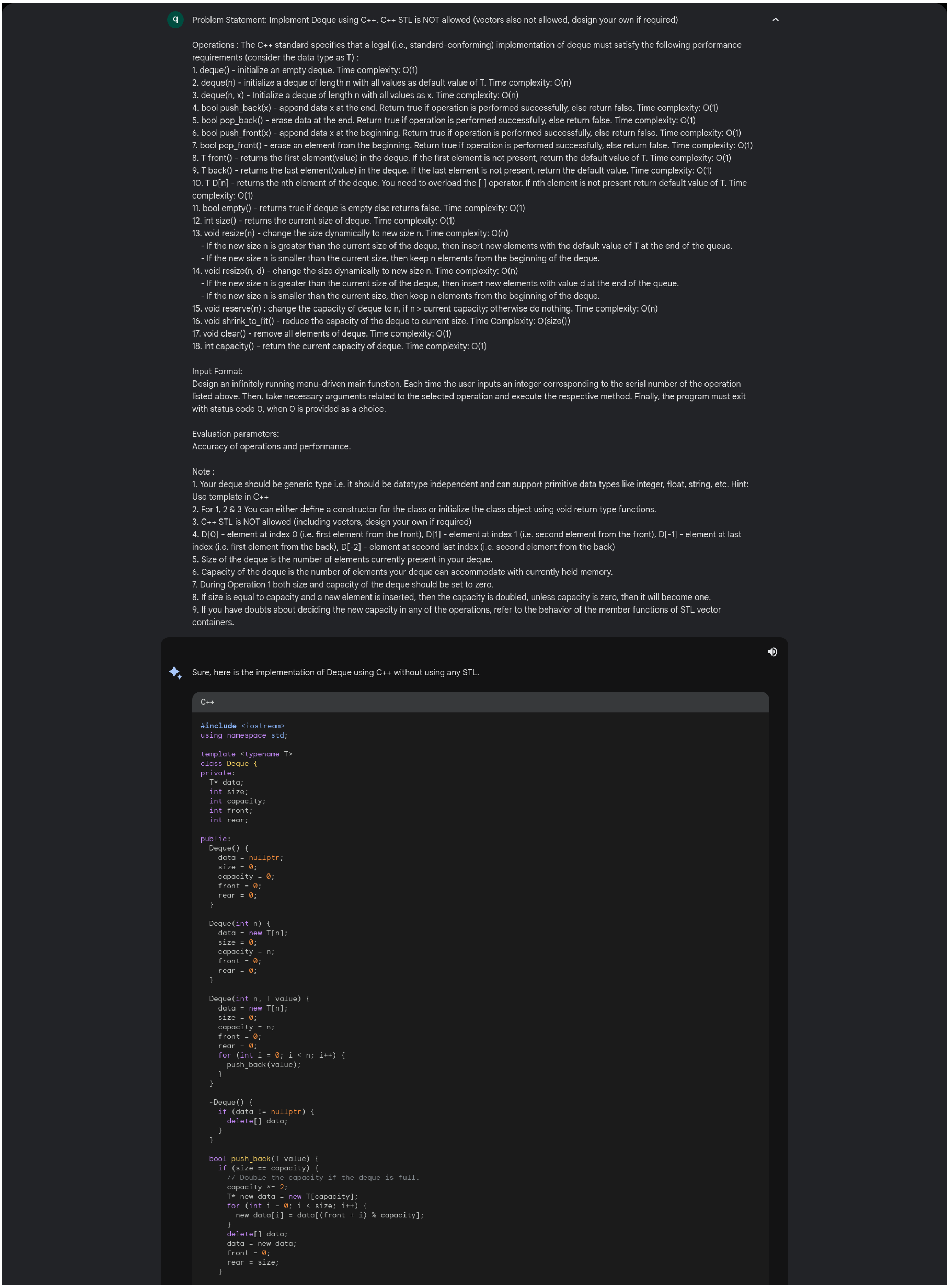


Problem Statement: Implement Deque using C++. C++ STL is NOT allowed (vectors also not allowed, design your own if required)

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.

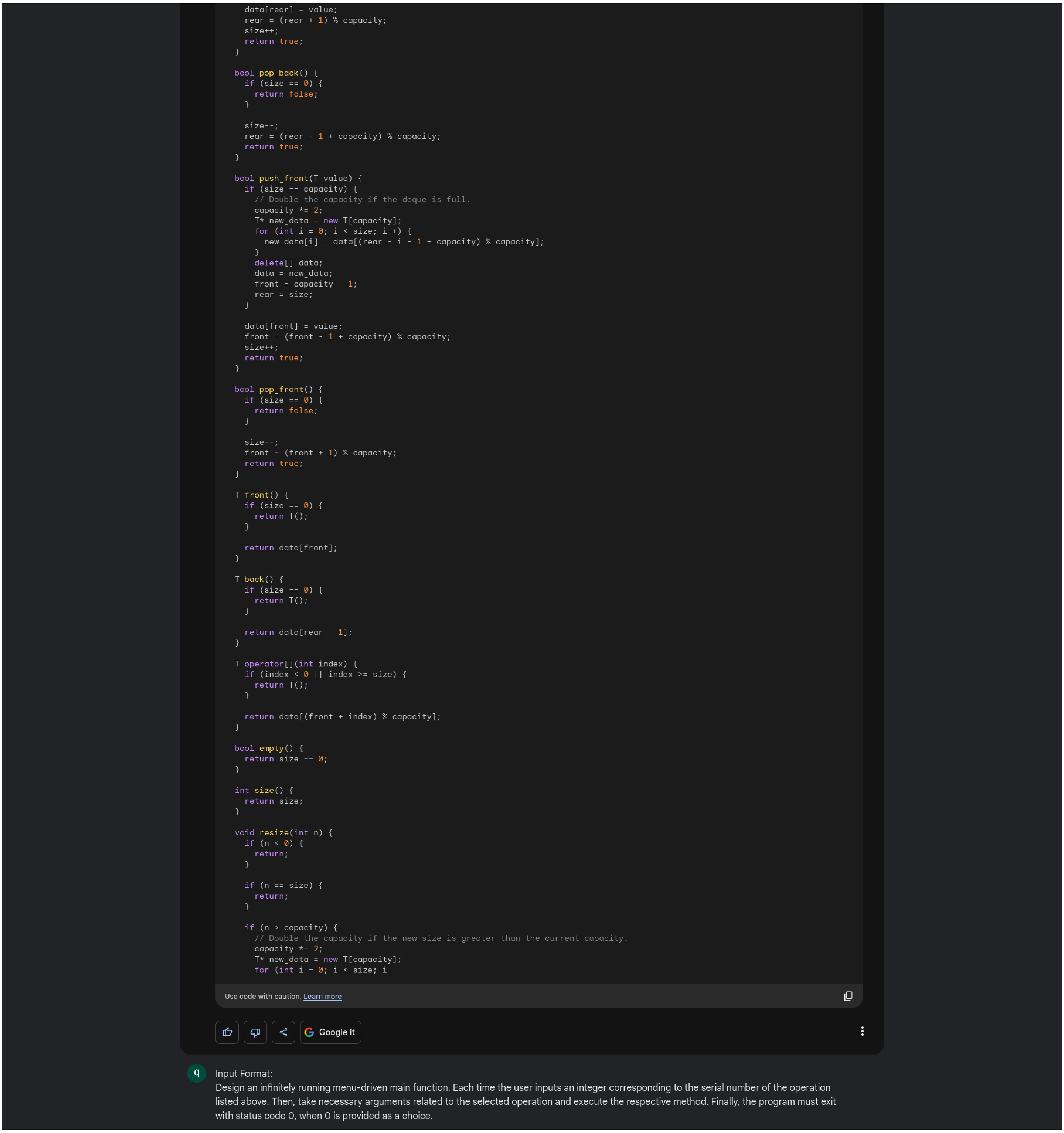
Evaluation parameters:
Accuracy of operations and performance.

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.

Sure, here is the implementation of Deque using C++ without using any STL.

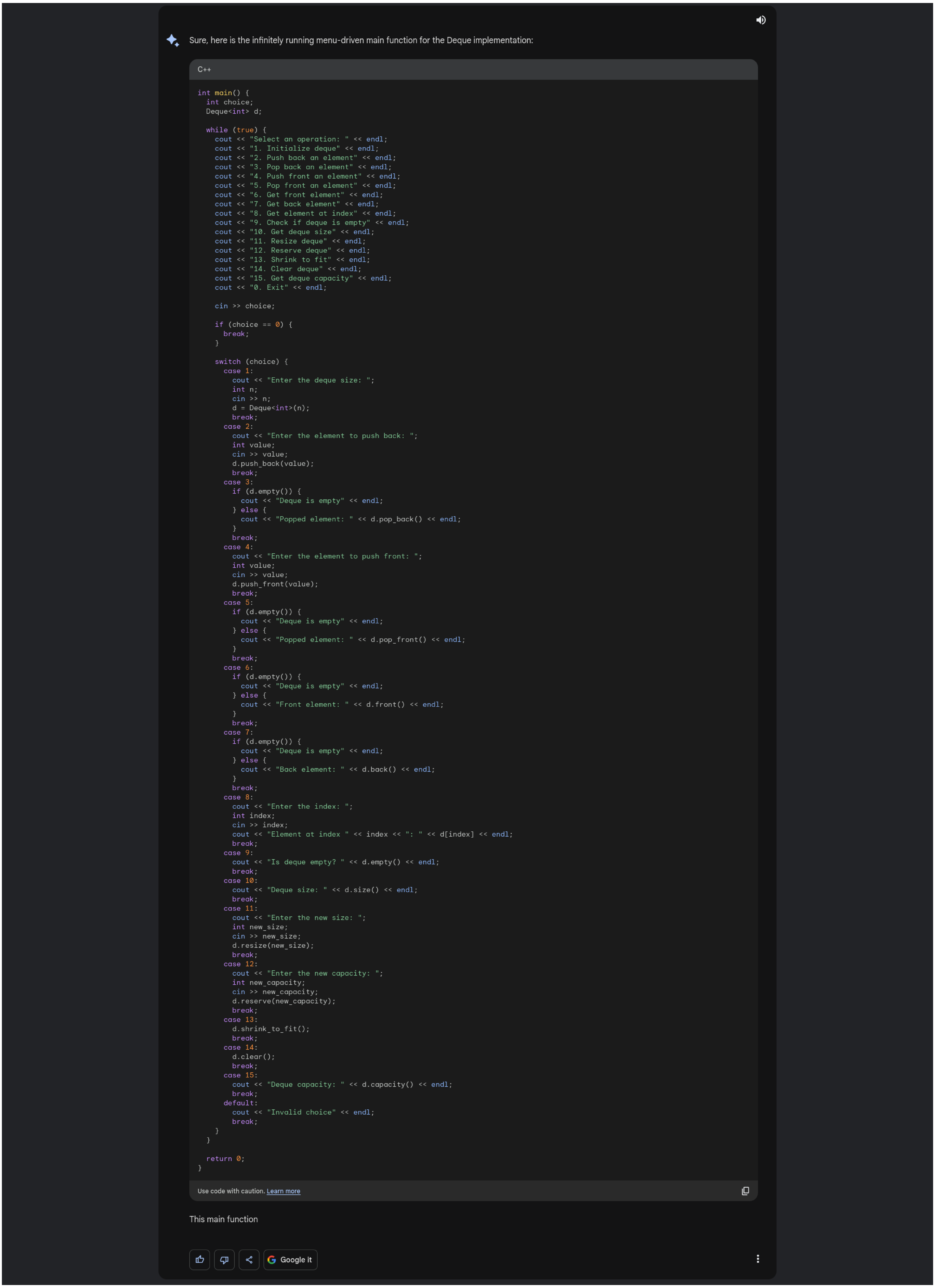
C++

```cpp
#include <iostream>
using namespace std;

template <typename T>
class Deque {
private:
  T* data;
  int size;
  int capacity;
  int front;
  int rear;

public:
  Deque() {
    data = nullptr;
    size = 0;
    capacity = 0;
    front = 0;
    rear = 0;
  }

  Deque(int n) {
    data = new T[n];
    size = 0;
    capacity = n;
    front = 0;
    rear = 0;
  }

  Deque(int n, T value) {
    data = new T[n];
    size = 0;
    capacity = n;
    front = 0;
    rear = 0;
    for (int i = 0; i < n; i++) {
      push_back(value);
    }
  }

  ~Deque() {
    if (data != nullptr) {
      delete[] data;
    }
  }

  bool push_back(T value) {
    if (size == capacity) {
      // Double the capacity if the deque is full.
      capacity *= 2;
      T* new_data = new T[capacity];
      for (int i = 0; i < size; i++) {
        new_data[i] = data[(front + i) % capacity];
      }
      delete[] data;
      data = new_data;
      front = 0;
      rear = size;
    }
```



```
    data[rear] = value;
    rear = (rear + 1) % capacity;
    size++;
    return true;
  }

  bool pop_back() {
    if (size == 0) {
      return false;
    }

    size--;
    rear = (rear - 1 + capacity) % capacity;
    return true;
  }

  bool push_front(T value) {
    if (size == capacity) {
      // Double the capacity if the deque is full.
      capacity *= 2;
      T* new_data = new T[capacity];
      for (int i = 0; i < size; i++) {
        new_data[i] = data[(rear - i - 1 + capacity) % capacity];
      }
      delete[] data;
      data = new_data;
      front = capacity - 1;
      rear = size;
    }

    data[front] = value;
    front = (front - 1 + capacity) % capacity;
    size++;
    return true;
  }

  bool pop_front() {
    if (size == 0) {
      return false;
    }

    size--;
    front = (front + 1) % capacity;
    return true;
  }

  T front() {
    if (size == 0) {
      return T();
    }

    return data[front];
  }

  T back() {
    if (size == 0) {
      return T();
    }

    return data[rear - 1];
  }

  T operator[](int index) {
    if (index < 0 || index >= size) {
      return T();
    }

    return data[(front + index) % capacity];
  }

  bool empty() {
    return size == 0;
  }

  int size() {
    return size;
  }

  void resize(int n) {
    if (n < 0) {
      return;
    }

    if (n == size) {
      return;
    }

    if (n > capacity) {
      // Double the capacity if the new size is greater than the current capacity.
      capacity *= 2;
      T* new_data = new T[capacity];
      for (int i = 0; i < size; i
```

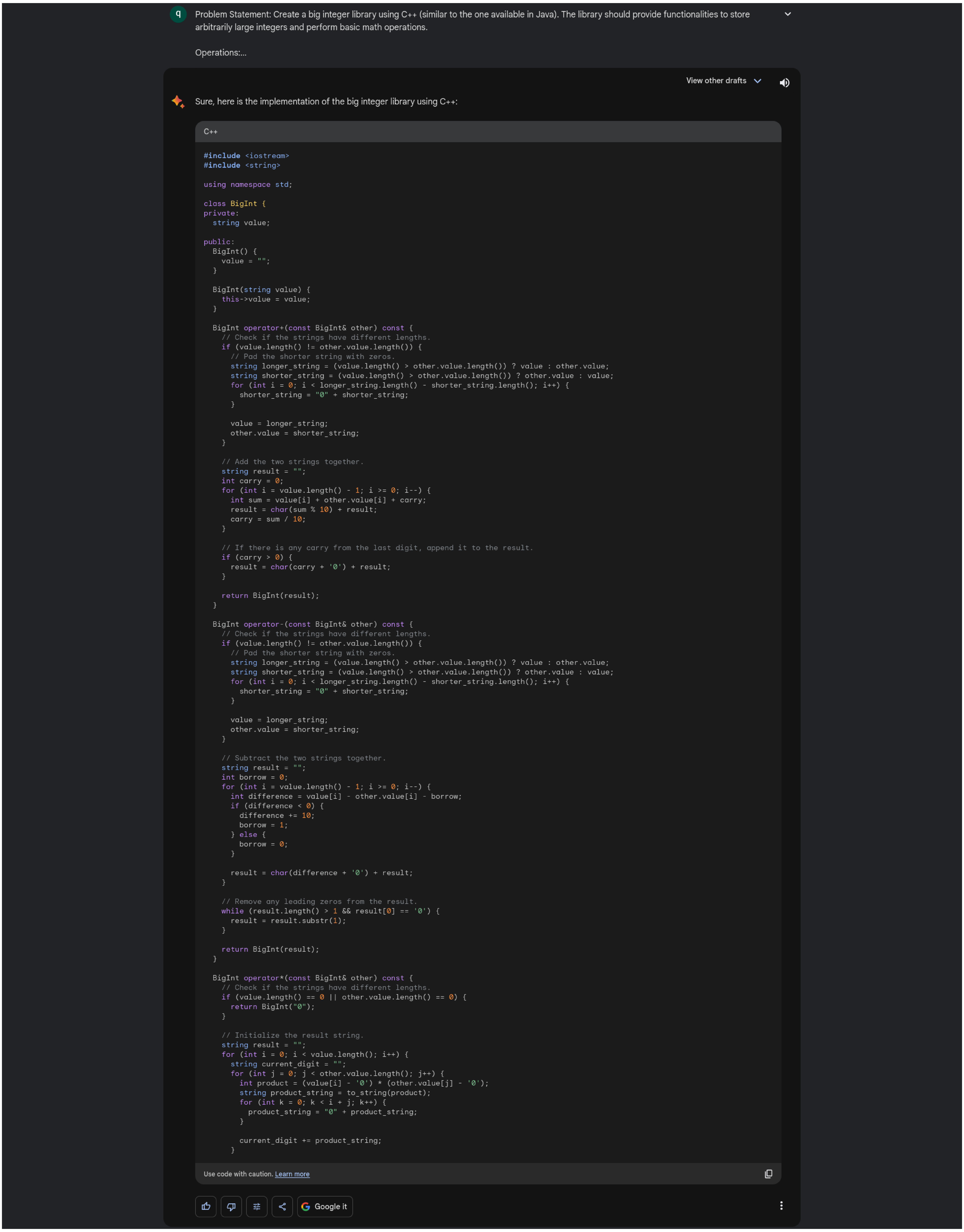
Input Format:
Design an infinitely running menu-driven main function. Each time the user inputs an integer corresponding to the serial number of the operation listed above. Then, take necessary arguments related to the selected operation and execute the respective method. Finally, the program must exit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



Sure, here is the infinitely running menu-driven main function for the Deque implementation:

```cpp
int main() {
  int choice;
  Deque<int> d;

  while (true) {
    cout << "Select an operation: " << endl;
    cout << "1. Initialize deque" << endl;
    cout << "2. Push back an element" << endl;
    cout << "3. Pop back an element" << endl;
    cout << "4. Push front an element" << endl;
    cout << "5. Pop front an element" << endl;
    cout << "6. Get front element" << endl;
    cout << "7. Get back element" << endl;
    cout << "8. Get element at index" << endl;
    cout << "9. Check if deque is empty" << endl;
    cout << "10. Get deque size" << endl;
    cout << "11. Resize deque" << endl;
    cout << "12. Reserve deque" << endl;
    cout << "13. Shrink to fit" << endl;
    cout << "14. Clear deque" << endl;
    cout << "15. Get deque capacity" << endl;
    cout << "0. Exit" << endl;

    cin >> choice;

    if (choice == 0) {
      break;
    }

    switch (choice) {
      case 1:
        cout << "Enter the deque size: ";
        int n;
        cin >> n;
        d = Deque<int>(n);
        break;
      case 2:
        cout << "Enter the element to push back: ";
        int value;
        cin >> value;
        d.push_back(value);
        break;
      case 3:
        if (d.empty()) {
          cout << "Deque is empty" << endl;
        } else {
          cout << "Popped element: " << d.pop_back() << endl;
        }
        break;
      case 4:
        cout << "Enter the element to push front: ";
        int value;
        cin >> value;
        d.push_front(value);
        break;
      case 5:
        if (d.empty()) {
          cout << "Deque is empty" << endl;
        } else {
          cout << "Popped element: " << d.pop_front() << endl;
        }
        break;
      case 6:
        if (d.empty()) {
          cout << "Deque is empty" << endl;
        } else {
          cout << "Front element: " << d.front() << endl;
        }
        break;
      case 7:
        if (d.empty()) {
          cout << "Deque is empty" << endl;
        } else {
          cout << "Back element: " << d.back() << endl;
        }
        break;
      case 8:
        cout << "Enter the index: ";
        int index;
        cin >> index;
        cout << "Element at index " << index << ": " << d[index] << endl;
        break;
      case 9:
        cout << "Is deque empty? " << d.empty() << endl;
        break;
      case 10:
        cout << "Deque size: " << d.size() << endl;
        break;
      case 11:
        cout << "Enter the new size: ";
        int new_size;
        cin >> new_size;
        d.resize(new_size);
        break;
      case 12:
        cout << "Enter the new capacity: ";
        int new_capacity;
        cin >> new_capacity;
        d.reserve(new_capacity);
        break;
      case 13:
        d.shrink_to_fit();
        break;
      case 14:
        d.clear();
        break;
      case 15:
        cout << "Deque capacity: " << d.capacity() << endl;
        break;
      default:
        cout << "Invalid choice" << endl;
        break;
    }
  }

  return 0;
}
```

This main function



Problem Statement: Create a big integer library using C++ (similar to the one available in Java). The library should provide functionalities to store arbitrarily large integers and perform basic math operations.

Operations:...

Sure, here is the implementation of the big integer library using C++:

```cpp
#include <iostream>
#include <string>

using namespace std;

class BigInt {
private:
  string value;

public:
  BigInt() {
    value = "";
  }

  BigInt(string value) {
    this->value = value;
  }

  BigInt operator+(const BigInt& other) const {
    // Check if the strings have different lengths.
    if (value.length() != other.value.length()) {
      // Pad the shorter string with zeros.
      string longer_string = (value.length() > other.value.length()) ? value : other.value;
      string shorter_string = (value.length() > other.value.length()) ? other.value : value;
      for (int i = 0; i < longer_string.length() - shorter_string.length(); i++) {
        shorter_string = "0" + shorter_string;
      }

      value = longer_string;
      other.value = shorter_string;
    }

    // Add the two strings together.
    string result = "";
    int carry = 0;
    for (int i = value.length() - 1; i >= 0; i--) {
      int sum = value[i] + other.value[i] + carry;
      result = char(sum % 10) + result;
      carry = sum / 10;
    }

    // If there is any carry from the last digit, append it to the result.
    if (carry > 0) {
      result = char(carry + '0') + result;
    }

    return BigInt(result);
  }

  BigInt operator-(const BigInt& other) const {
    // Check if the strings have different lengths.
    if (value.length() != other.value.length()) {
      // Pad the shorter string with zeros.
      string longer_string = (value.length() > other.value.length()) ? value : other.value;
      string shorter_string = (value.length() > other.value.length()) ? other.value : value;
      for (int i = 0; i < longer_string.length() - shorter_string.length(); i++) {
        shorter_string = "0" + shorter_string;
      }

      value = longer_string;
      other.value = shorter_string;
    }

    // Subtract the two strings together.
    string result = "";
    int borrow = 0;
    for (int i = value.length() - 1; i >= 0; i--) {
      int difference = value[i] - other.value[i] - borrow;
      if (difference < 0) {
        difference += 10;
        borrow = 1;
      } else {
        borrow = 0;
      }

      result = char(difference + '0') + result;
    }

    // Remove any leading zeros from the result.
    while (result.length() > 1 && result[0] == '0') {
      result = result.substr(1);
    }

    return BigInt(result);
  }

  BigInt operator*(const BigInt& other) const {
    // Check if the strings have different lengths.
    if (value.length() == 0 || other.value.length() == 0) {
      return BigInt("0");
    }

    // Initialize the result string.
    string result = "";
    for (int i = 0; i < value.length(); i++) {
      string current_digit = "";
      for (int j = 0; j < other.value.length(); j++) {
        int product = (value[i] - '0') * (other.value[j] - '0');
        string product_string = to_string(product);
        for (int k = 0; k < i + j; k++) {
          product_string = "0" + product_string;
        }

        current_digit += product_string;
      }
```

Use code with caution. Learn more