

VECTORS :-

```
#include <iostream>
```

```
#include <vector>
```

```
class VectorPair {
```

```
private:
```

```
    std::vector<int> vector1;
```

```
    std::vector<int> vector2;
```

```
public:
```

```
    VectorPair(const std::vector<int>& v1, const std::vector<int>& v2) : vector1(v1), vector2(v2) {}
```

```
// Constructor to initialize the vectors with given sizes and values
```

```
    void addElementsToVector1(int value) {                                // Method to add elements  
to the vectors
```

```
        vector1.push_back(value);
```

```
    }
```

```
    void addElementsToVector2(int value) {
```

```
        vector2.push_back(value);
```

```
    }
```

```
    void displayVectors() const {                                        // Method to display the  
vectors
```

```
        std::cout << "Vector 1: ";
```

```
        for (const auto& elem : vector1) {
```

```
            std::cout << elem << " ";
```

```
        }
```

```
        std::cout << std::endl;
```

```
        std::cout << "Vector 2: ";
```

```
        for (const auto& elem : vector2) {
```

```

        std::cout << elem << " ";

    }

    std::cout << std::endl;
}

size_t getVector1Size() const {                // Method to get the size of the vectors
    return vector1.size();
}

size_t getVector2Size() const {
    return vector2.size();
}

void clearVectors() {                          // Method to clear the vectors
    vector1.clear();
    vector2.clear();
}

};

int main() {
    std::vector<int> v1 = {1, 2, 3};
    std::vector<int> v2 = {4, 5, 6};
    VectorPair vp(v1, v2);
    vp.displayVectors();
    vp.addElementToVector1(7);
    vp.addElementToVector2(8);
    vp.displayVectors();
    std::cout << "Size of Vector 1: " << vp.getVector1Size() << std::endl;
}

```

```

std::cout << "Size of Vector 2: " << vp.getVector2Size() << std::endl;

vp.clearVectors();

vp.displayVectors();

return 0;

}

```

OUTPUT :-

The screenshot shows an online C++ compiler interface. The top part displays the source code for a program that uses two vectors, Vector1 and Vector2, and a VectorPair class. The code includes headers for <iostream> and <vector>, defines the VectorPair class with methods for adding elements, displaying vectors, and getting their sizes, and then uses these in the main function. The bottom part shows the output of the program, which prints the contents of both vectors and their sizes. The output is as follows:

```

Vector 1: 1 2 3 7
Vector 2: 4 5 6 8
Size of Vector 1: 4
Size of Vector 2: 4
Vector 1:
Vector 2:
...Program finished with exit code 0
Press ENTER to exit console.

```

QUEUE::PUSH() :-

```

#include <iostream>

#include <queue>

using namespace std;

int main()

{

    queue<int> myqueue;

    myqueue.push(0);

    myqueue.push(1);

```

```

myqueue.push(2);
while(!myqueue.empty()) {
    cout << ' ' << myqueue.front();
    myqueue.pop();
}
}

```

OUTPUT :-

The screenshot shows a web browser window with an online C++ compiler. The code in the editor is as follows:

```

1 #include <iostream>
2 #include <queue>
3 using namespace std;
4 int main()
5 {
6     queue<int> myqueue;
7     myqueue.push(0);
8     myqueue.push(1);
9     myqueue.push(2);
10    while(!myqueue.empty()) {
11        cout << ' ' << myqueue.front();
12        myqueue.pop();
13    }
14 }

```

The output window shows the result of the program execution:

```

0 1 2
...Program finished with exit code 0
Press ENTER to exit console.

```

The Windows taskbar at the bottom shows the system time as 12:02 on 10-07-2024.

QUEUE:: POP() :-

```

#include <iostream>
#include <queue>
using namespace std;
int main()
{
    queue<int> myqueue;

    myqueue.push(0);
    myqueue.push(1);

```

```

myqueue.push(2);

myqueue.pop();

myqueue.pop();

while(!myqueue.empty()) {

    cout << ' ' << myqueue.front();

    myqueue.pop();

}

}

```

OUTPUT :-

The screenshot shows a web browser window with the URL https://www.onlinegdb.com/online_c_compiler#. The code editor contains the following C++ code:

```

1 #include <iostream>
2 #include <queue>
3 using namespace std;
4 int main()
5 {
6     queue<int> myqueue;
7     myqueue.push(1);
8     myqueue.push(2);
9     myqueue.push(3);
10    myqueue.pop();
11    myqueue.pop();
12    while( myqueue.empty() ) {
13        cout << ' ' << myqueue.front();
14        myqueue.pop();
15    }
16 }

```

The output window shows the following text:

```

2
...Program finished with exit code 0
Press ENTER to exit console.

```

STACK CONTINUED :-

```

#include <bits/stdc++.h>

using namespace std;

void showstack(stack <int> s)

{

    while(!s.empty())

    {

```

```

        cout << '\t' << s.top();

        s.pop();

    }

    cout << '\n';

}

int main()

{

    stack <int> s;

    s.push(10);

    s.push(30);

    s.push(20);

    s.push(5);

    s.push(1);

    cout << "The stack is : ";

    showstack(s);

    cout << "\ns.size() : " << s.size();

    cout << "\ns.top() : " << s.size();

    cout << "\ns.pop() : ";

    s.pop();

    showstack(s);

    return 0;

}

```

OUTPUT :

The screenshot shows a web browser with the URL https://www.onlinegdb.com/online_c_compiler#. The code editor contains the following C++ code:

```
1 #include <iostream>
2 using namespace std;
3 void showstack(stack<int> s)
4 {
5     while(!s.empty())
6     {
7         cout << '\t' << s.top();
8         s.pop();
9     }
10    cout << '\n';
11 }
12 int main()
13 {
14     stack<int> s;
15     s.push(10);
16     s.push(20);
17     s.push(30);
18     s.push(40);
19     s.push(50);
20     cout << "The stack is : ";
21     showstack(s);
22     cout << "s.size() : " << s.size();
23     cout << "s.top() : " << s.top();
24     cout << "s.pop() : ";
25     s.pop();
26     showstack(s);
27     return 0;
28 }
```

The output window shows the following results:

```
The stack is : 1    5    20    30    10
s.size() : 5
s.top() : 5
s.pop() : 5    20    30    10
...Program finished with exit code 0
Press ENTER to exit console.
```

Problem :-Reverse a Queue

Description:

Implement a function to reverse the elements of a queue using a stack.

```
#include <iostream>
```

```
#include <queue>
```

```
#include <stack>
```

```
using namespace std;
```

```
void reverseQueue(queue<int> &q) {
```

```
    stack<int> s;
```

```
    while (!q.empty()) {                                // Step 1: Push all elements from queue to stack
```

```
        s.push(q.front());
```

```
        q.pop();
```

```
    }
```

```
    while (!s.empty()) {                                // Step 2: Pop all elements from stack back to queue
```

```
        q.push(s.top());
```

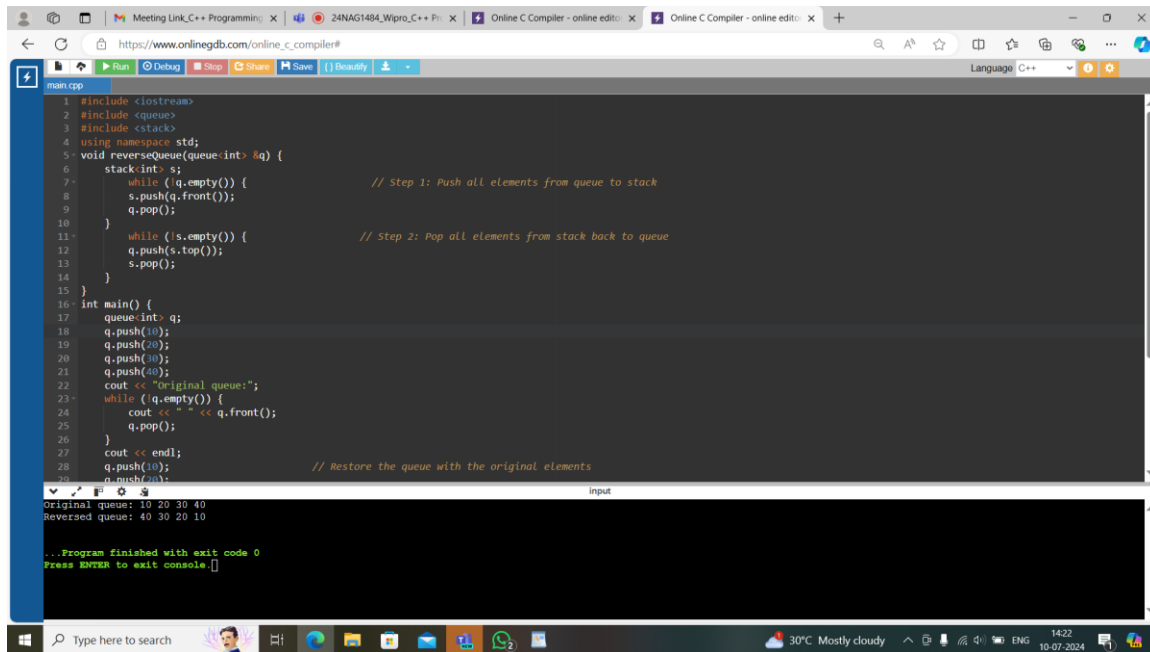
```

        s.pop();
    }
}

int main() {
    queue<int> q;
    q.push(10);
    q.push(20);
    q.push(30);
    q.push(40);
    cout << "Original queue:";
    while (!q.empty()) {
        cout << " " << q.front();
        q.pop();
    }
    cout << endl;
    q.push(10);                // Restore the queue with the original elements
    q.push(20);
    q.push(30);
    q.push(40);
    reverseQueue(q);          // Reverse the queue
    cout << "Reversed queue:";
    while (!q.empty()) {
        cout << " " << q.front();
        q.pop();
    }
    cout << endl;
    return 0;
}

```


OUTPUT :-



```
1 #include <iostream>
2 #include <queue>
3 #include <stack>
4 using namespace std;
5 void reverseQueue(queue<int> &q) {
6     stack<int> s;
7     while (!q.empty()) {           // Step 1: Push all elements from queue to stack
8         s.push(q.front());
9         q.pop();
10    }
11    while (!s.empty()) {           // Step 2: Pop all elements from stack back to queue
12        q.push(s.top());
13        s.pop();
14    }
15 }
16 int main() {
17     queue<int> q;
18     q.push(10);
19     q.push(20);
20     q.push(30);
21     q.push(40);
22     cout << "Original queue:";
23     while (!q.empty()) {
24         cout << " " << q.front();
25         q.pop();
26     }
27     cout << endl;
28     q.push(10);           // Restore the queue with the original elements
29     q.push(20);
30     q.push(30);
31     q.push(40);
32 }
```

Original queue: 10 20 30 40
Reversed queue: 40 30 20 10

... Program finished with exit code 0
Press ENTER to exit console

Maximum Element in Stack

Description:

Design a stack that supports push, pop, and retrieving the maximum element in constant time.

```
#include <iostream>
```

```
#include <stack>
```

```
class Queue {
```

```
private:
```

```
    std::stack<int> inbox;           // For enqueue operation
```

```
    std::stack<int> outbox;          // For dequeue operation
```

```
public:
```

```
    void enqueue(int x) {
```

```
        inbox.push(x);
```

```
    }
```

```
    int dequeue() {
```

```
        if (outbox.empty()) {
```

```

        while (!inbox.empty()) {
            from inbox to outbox
            outbox.push(inbox.top());
            inbox.pop();
        }
    }

    if (outbox.empty()) {
        std::cerr << "Error: Queue is empty!" << std::endl;
        return -1;
    }

    int front = outbox.top();
    outbox.pop();
    return front;
}

bool empty() const {
    return inbox.empty() && outbox.empty();
}

};

int main() {
    Queue q;
    q.enqueue(1);
    q.enqueue(2);
    q.enqueue(3);
    std::cout << q.dequeue() << std::endl;    // Output: 1
    std::cout << q.dequeue() << std::endl;    // Output: 2
    q.enqueue(4);
    std::cout << q.dequeue() << std::endl;    // Output: 3
    std::cout << q.dequeue() << std::endl;    // Output: 4
}

```

```

std::cout << q.dequeue() << std::endl; // Output: Error message, queue is empty

return 0;

}

```

OUTPUT :-

```

1 #include <iostream>
2 #include <stack>
3 class Queue {
4 private:
5     std::stack<int> inbox;           // For enqueue operation
6     std::stack<int> outbox;          // For dequeue operation
7 public:
8     void enqueue(int x) {
9         inbox.push(x);
10    }
11    int dequeue() {
12        if (outbox.empty()) {
13            while (!inbox.empty()) { // Transfer elements from inbox to outbox
14                outbox.push(inbox.top());
15                inbox.pop();
16            }
17        }
18        if (outbox.empty()) {
19            std::cerr << "Error: Queue is empty!" << std::endl;
20            return -1; // Error case, queue is empty
21        }
22        int front = outbox.top();
23        outbox.pop();
24        return front;
25    }
26    bool empty() const {
27        return inbox.empty() && outbox.empty();
28    }
29 };
30
31 int main() {
32     Queue q;
33     q.enqueue(1);
34     q.enqueue(2);
35     q.enqueue(3);
36     q.enqueue(4);
37     q.dequeue();
38     q.dequeue();
39     q.dequeue();
40     q.dequeue();
41     q.dequeue();
42     return 0;
43 }

```

Output:

```

1
2
3
4
Error: Queue is empty!
-1
...Program finished with exit code 0
Press ENTER to exit console.

```

Circular Queue Implementation

Description:

Implement a circular queue using an array. The queue should support enqueue, dequeue, and front operations.

```
#include <iostream>
```

```
using namespace std;
```

```
class CircularQueue {
```

```
private:
```

```
    int *arr;
```

```
    int front;
```

```
    int rear;
```

```
    int capacity;
```

```

int size;

public:

    CircularQueue(int cap) { // Constructor to
initialize the queue

        capacity = cap;

        arr = new int[capacity];

        front = 0;

        rear = -1;

        size = 0;

    }

    ~CircularQueue() { // Destructor to free dynamically
allocated memory

        delete[] arr;

    }

    bool isEmpty() { // Function to check
if queue is empty

        return size == 0;

    }

    bool isFull() { // Function to check if
queue is full

        return size == capacity;

    }

    int getSize() { // Function to
get the current size of the queue

        return size;

    }

    void enqueue(int value) { // Function to
enqueue an element to the queue

        if (isFull()) {

            cout << "Queue Overflow\n";

```

```

        return;
    }

    rear = (rear + 1) % capacity;
    arr[rear] = value;
    size++;

    cout << value << " enqueued to queue\n";
}

void dequeue() { //
Function to dequeue an element from the queue

    if (isEmpty()) {
        cout << "Queue Underflow\n";
        return;
    }

    int removedValue = arr[front];
    front = (front + 1) % capacity;
    size--;

    cout << removedValue << " dequeued from queue\n";
}

int frontElement() { // Function to
get the front element of the queue

    if (isEmpty()) {
        cout << "Queue is empty\n";
        return -1;
    }

    return arr[front];
}

```

```
};  
  
int main() {  
    CircularQueue queue(5);  
    queue.enqueue(1);  
    queue.enqueue(2);  
    queue.enqueue(3);  
    queue.enqueue(4);  
    queue.enqueue(5);  
    cout << "Front element: " << queue.frontElement() << endl;  
    queue.dequeue();  
    queue.dequeue();  
    cout << "Front element after dequeuing: " << queue.frontElement() << endl;  
    queue.enqueue(6);  
    queue.enqueue(7);  
    queue.dequeue();  
    queue.dequeue();  
    queue.dequeue();  
    queue.dequeue();  
    queue.dequeue();  
    return 0;  
}
```

OUTPUT :-

The screenshot shows a web browser with multiple tabs of an online C++ compiler. The main editor displays the following C++ code for a CircularQueue:

```
1 #include <iostream>
2 using namespace std;
3 class CircularQueue {
4 private:
5     int *arr;
6     int front;
7     int rear;
8     int capacity;
9     int size;
10 public:
11     CircularQueue(int cap) { // constructor to initialize the queue
12         capacity = cap;
13         arr = new int[capacity];
14         front = 0;
15         rear = -1;
16         size = 0;
17     }
18     ~CircularQueue() { // Destructor to free dynamically allocated memory
19         delete[] arr;
20     }
21     bool isEmpty() { // Function to check if queue is empty
22         return size == 0;
23     }
24     bool isFull() { // Function to check if queue is full
25         return size == capacity;
26     }
27     int getSize() { // Function to get the current size of the queue
28         return size;
29     }
30 }
```

Below the code editor, the output window shows the following execution results:

```
1 enqueued to queue
2 enqueued to queue
3 enqueued to queue
4 enqueued to queue
5 enqueued to queue
Front element: 1
1 dequeued from queue
2 dequeued from queue
Front element after dequeuing: 3
6 enqueued to queue
```

Sort a Stack

Description:

Write a function to sort a stack such that the smallest items are on the top.

```
#include <iostream>
```

```
#include <stack>
```

```
using namespace std;
```

```
class Queue {
```

```
private:
```

```
    stack<int> inbox; // For enqueue operation
```

```
    stack<int> outbox; // For dequeue operation
```

```
public:
```

```
    void enqueue(int x) {
```

```
        inbox.push(x);
```

```
    }
```

```
    int dequeue() {
```

```
        if (outbox.empty()) {
```

```

        while (!inbox.empty()) {                                // Transfer all
elements from inbox to outbox

            outbox.push(inbox.top());

            inbox.pop();

        }

    }

    if (!outbox.empty()) {                                       // Pop the front
element from outbox (if exists)

        int front = outbox.top();

        outbox.pop();

        return front;

    } else {

        cout << "Queue is empty!" << endl;

        return -1; // Or throw an exception

    }

}

};

int main() {

    Queue q;

    q.enqueue(1);                                                // Enqueue some elements

    q.enqueue(2);

    q.enqueue(3);

    cout << q.dequeue() << endl;                                // Dequeue elements

    cout << q.dequeue() << endl;

    q.enqueue(4);                                                // Enqueue more elements

    q.enqueue(5);

    cout << q.dequeue() << endl; // Output: 3                    // Dequeue remaining
elements

    cout << q.dequeue() << endl; // Output: 4

```



```

cout << q.dequeue() << endl; // Output: 5

cout << q.dequeue() << endl; // Output: Queue is empty!

return 0;

}

```

OUTPUT :-

The screenshot shows a web browser window with an online C++ compiler. The code in the editor implements a queue using two stacks, 'inbox' and 'outbox'. The 'enqueue' function pushes elements into 'inbox'. The 'dequeue' function transfers elements from 'inbox' to 'outbox' until 'inbox' is empty, then pops from 'outbox'. If 'outbox' is empty, it prints 'Queue is empty!' and returns -1. The main function tests the queue with inputs 10, 20, 30, 5, and 1, producing the output shown in the console.

```

main.cpp
1 #include <iostream>
2 #include <stack>
3 using namespace std;
4 class Queue {
5 private:
6     stack<int> inbox;    // For enqueue operation
7     stack<int> outbox;   // For dequeue operation
8 public:
9     void enqueue(int x) {
10         inbox.push(x);
11     }
12     int dequeue() {
13         if (outbox.empty()) {
14             while (!inbox.empty()) {
15                 outbox.push(inbox.top());
16                 inbox.pop();
17             }
18             // Transfer all elements from inbox to outbox
19             if (outbox.empty()) {
20                 int front = outbox.top();
21                 outbox.pop();
22                 return front;
23             }
24             // Pop the front element from outbox (if exists)
25             cout << "Queue is empty!" << endl;
26             return -1; // Or throw an exception
27         }
28     };
29 };
30 int main() {
31     Queue q;
32     q.enqueue(10);
33     q.enqueue(20);
34     q.enqueue(30);
35     q.dequeue();
36     q.dequeue();
37     q.dequeue();
38     q.dequeue();
39     q.dequeue();
40     return 0;
41 }

```

Output in console:

```

1
2
3
4
5
Queue is empty!
-1
...Program finished with exit code 0

```

STD::LIST :-

```
#include <iostream>
```

```
#include <list>
```

```
int main() {
```

```
    std::list<int> myList; // Create a list
```

```
    myList.push_back(10); // Insert elements at the end
```

```
    myList.push_back(20);
```

```
    myList.push_back(30);
```

```
    myList.push_front(5); // Insert elements at the front
```

```
    myList.push_front(1);
```

```

std::cout << "List after push_back and push_front: ";           // Display elements
for (int val : myList) {
    std::cout << val << " ";
}
std::cout << std::endl;
auto it = myList.begin();                                       // Insert element at a specific position
std::advance(it, 2);
myList.insert(it, 15);
std::cout << "List after insert: ";
for (int val : myList) {
    std::cout << val << " ";
}
std::cout << std::endl;
it = myList.begin();                                           // Erase element at a specific position
std::advance(it, 3);
myList.erase(it);
std::cout << "List after erase: ";
for (int val : myList) {
    std::cout << val << " ";
}
std::cout << std::endl;
myList.remove(10);                                             // Remove elements by value
std::cout << "List after remove: ";
for (int val : myList) {
    std::cout << val << " ";
}
std::cout << std::endl;
myList.remove_if([](int n) { return n < 10; });               // Remove elements based on a

```

condition

```
std::cout << "List after remove_if: ";
for (int val : myList) {
    std::cout << val << " ";
}
std::cout << std::endl;

myList.sort(); // Sorting the list

std::cout << "List after sort: ";
for (int val : myList) {
    std::cout << val << " ";
}
std::cout << std::endl;

myList.reverse(); // Reversing the list

std::cout << "List after reverse: ";
for (int val : myList) {
    std::cout << val << " ";
}
std::cout << std::endl;

std::list<int> otherList = {40, 50, 60}; // Merging two lists
myList.merge(otherList);

std::cout << "List after merge: ";
for (int val : myList) {
    std::cout << val << " ";
}
std::cout << std::endl;

myList.clear(); // Clearing the list

std::cout << "List after clear: ";
for (int val : myList) {
```

```

        std::cout << val << " ";
    }

    std::cout << std::endl;

    if (myList.empty()) {                                // Checking if the list is empty

        std::cout << "List is empty." << std::endl;
    }

    myList.push_back(100);                                // Adding elements again
    myList.push_back(200);

    std::cout << "Front element: " << myList.front() << std::endl;    // Accessing front and
back elements
    std::cout << "Back element: " << myList.back() << std::endl;

    return 0;
}

```

OUTPUT :-

The screenshot shows a web browser window with the URL https://www.onlinegdb.com/online_c_compiler#. The code editor contains the following C++ code:

```

1 #include <iostream>
2 #include <list>
3 int main() {
4     std::list<int> myList;           // create a list
5     myList.push_back(10);           // Insert elements at the end
6     myList.push_back(20);
7     myList.push_back(30);
8
9     myList.push_front(40);           // Insert elements at the front
10    myList.push_front(50);
11    std::cout << "List after push_back and push_front: ";           // Display elements
12    for (int val : myList) {
13        std::cout << val << " ";
14    }
15    std::cout << std::endl;
16    auto it = myList.begin();         // Insert element at a specific position
17    std::advance(it, 2);
18    myList.insert(it, 15);
19    std::cout << "List after insert: ";
20    for (int val : myList) {
21        std::cout << val << " ";
22    }
23    std::cout << std::endl;
24    it = myList.begin();              // Erase element at a specific position
25    std::advance(it, 3);
26    myList.erase(it);
27    std::cout << "List after erase: ";
28    for (int val : myList) {
29        std::cout << val << " ";
30    }
31    std::cout << std::endl;
32    myList.reverse();
33    std::cout << "List after reverse: ";
34    for (int val : myList) {
35        std::cout << val << " ";
36    }
37    std::cout << std::endl;
38    myList.clear();
39    std::cout << "List after clear: ";
40    for (int val : myList) {
41        std::cout << val << " ";
42    }
43    std::cout << std::endl;
44    std::cout << "List is empty." << std::endl;
45    return 0;
46 }

```

The output window shows the following results:

```

List after push_back and push_front: 1 5 10 20 30
List after insert: 1 5 15 10 20 30
List after erase: 1 5 15 20 30
List after reverse: 1 5 15 20 30
List after clear: 15 20 30
List after reverse: 30 20 15
List after merge: 30 20 15 40 50 60
List after clear:
List is empty.

```

