#1

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

  Q.push(5);
  Q.push(3);

  cout << "Front element: " << Q.front() << endl;
  Q.pop();
  cout << "Is queue empty? " << (Q.empty() ? "Yes" : "No")<< endl;
  cout << "Queue size: " << Q.size() << endl;
  return 0;
}
```

```
Front element: 5
Is queue empty? No
Queue size: 1
```

#2

```
#include <iostream>
#include <deque>
using namespace std;
int main() {
deque<int> dq;

  dq.push_back(10);
  dq.push_front(5);

  cout << "Front: " << dq.front() << endl;
  cout << "Back: " << dq.back() << endl;
  dq.pop_front();
  dq.pop_back();

  if (dq.empty()) {
  cout << "Deque is empty." << endl;
  }
  return 0;
}
```

```
Front: 5
Back: 10
Deque is empty.
```

#3

```
#include <iostream>
using namespace std;
#define MAX 1000
class Deque {
private:
int* arr;
int capacity;
int front;
int rear;
int currentSize;
public:

Deque(int cap) {
capacity = cap;
arr = new int[capacity];
front = -1;
rear = -1;
currentSize = 0;
}

void push_front(int e) {
if (currentSize == capacity) {
cout << "Deque Overflow" << endl;
return;
}
if (front == -1) {
front = 0;
rear = 0;
} else {

front = (front - 1 + capacity) % capacity;
}
arr[front] = e;
currentSize++;
}

void push_back(int e) {
```

```
if (currentSize == capacity) {
cout << "Deque Overflow" << endl;
return;
}
if (rear == -1) {
front = 0;
rear = 0;
} else {

rear = (rear + 1) % capacity;
}
arr[rear] = e;
currentSize++;
}

void pop_front() {
if (currentSize == 0) {
cout << "Deque Underflow" << endl;
return;
}
if (front == rear) {
front = -1;
rear = -1;
} else {

front = (front + 1) % capacity;
}
currentSize--;
}

void pop_back() {
if (currentSize == 0) {
cout << "Deque Underflow" << endl;
return;
}
if (front == rear) {
front = -1;
rear = -1;
} else {

rear = (rear - 1 + capacity) % capacity;
}
currentSize--;
}
```

```
int front_element() {
if (currentSize == 0) {
cout << "Deque is Empty" << endl;
return -1;
}
return arr[front];
}

int back_element() {
if (currentSize == 0) {
cout << "Deque is Empty" << endl;
return -1;
}
return arr[rear];
}

int size() {
return currentSize;
}

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

~Deque() {
delete[] arr;
}
};
int main() {

Deque dq(5);

dq.push_back(10);
dq.push_front(5);
dq.push_back(15);
dq.push_front(2);
dq.push_back(20);

dq.push_back(25);
cout << "Front element: " << dq.front_element() << endl;
cout << "Back element: " << dq.back_element() << endl;
dq.pop_front();
dq.pop_back();
```

```
cout << "Front element after pop: " << dq.front_element() << endl;
cout << "Back element after pop: " << dq.back_element() << endl;
cout << "Deque is " << (dq.empty() ? "empty" : "not empty") << endl;
return 0;
}
```

```
Deque Overflow
Front element: 2
Back element: 20
Front element after pop: 5
Back element after pop: 15
Deque is not empty
```

#4

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

struct node {
int data;
node* next;
node* prev;
};

class Deque {
private:
node* head;
node* tail;
int currentSize;
public:

Deque() {
head = nullptr;
tail = nullptr;
currentSize = 0;
}

void push_front(int e) {
node* newNode = new node;
newNode->data = e;
newNode->prev = nullptr;
newNode->next = head;
if (head == nullptr) {
head = newNode;
tail = newNode;
```

```
} else {
head->prev = newNode;
head = newNode;
}
currentSize++;
}

void push_back(int e) {
node* newNode = new node;
newNode->data = e;
newNode->next = nullptr;
newNode->prev = tail;
if (tail == nullptr) {
head = newNode;
tail = newNode;
} else {
tail->next = newNode;
tail = newNode;
}
currentSize++;
}
\
void pop_front() {
if (head == nullptr) {
cout << "Deque Underflow" << endl;
return;
}
node* temp = head;
head = head->next;
if (head != nullptr) {
head->prev = nullptr;
} else {
tail = nullptr;
}
delete temp;
currentSize--;
}

void pop_back() {
if (tail == nullptr) {
cout << "Deque Underflow" << endl;
return;
}
node* temp = tail;
```

```
tail = tail->prev;
if (tail != nullptr) {
tail->next = nullptr;
} else {
head = nullptr;
}
delete temp;
currentSize--;
}

int front() {
if (head == nullptr) {
cout << "Deque is Empty" << endl;
return -1;
}
return head->data;
}

int back() {
if (tail == nullptr) {
cout << "Deque is Empty" << endl;
return -1;
}
return tail->data;
}

int size() {
return currentSize;
}

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

~Deque() {
while (head != nullptr) {
pop_front();
}
}
};
int main() {

Deque dq;
```

```
dq.push_back(10);
dq.push_front(5);
dq.push_back(15);
dq.push_front(2);

cout << "Front element: " << dq.front() << endl;
cout << "Back element: " << dq.back() << endl;
dq.pop_front();
dq.pop_back();
cout << "Front element after pop: " << dq.front() << endl;
cout << "Back element after pop: " << dq.back() << endl;
cout << "Size of deque: " << dq.size() << endl;
cout << "Deque is " << (dq.empty() ? "empty" : "not empty") << endl;
return 0;
}
```

```
Front element: 2
Back element: 15
Front element after pop: 5
Back element after pop: 10
Size of deque: 2
Deque is not empty
```