

| | | | | | |
|--------------|-------------|----|------------|------|---------|
| 과목명 | 자료구조 및 알고리즘 | 분반 | x | 담당교수 | 김화성 교수님 |
| 학과 | 전자통신공학과 | 학번 | 2016707079 | 이름 | 하상천 |
| Queue ADT 구현 | | | | | |

1, 과제설명 (과제에 대한 설명 및 목표)

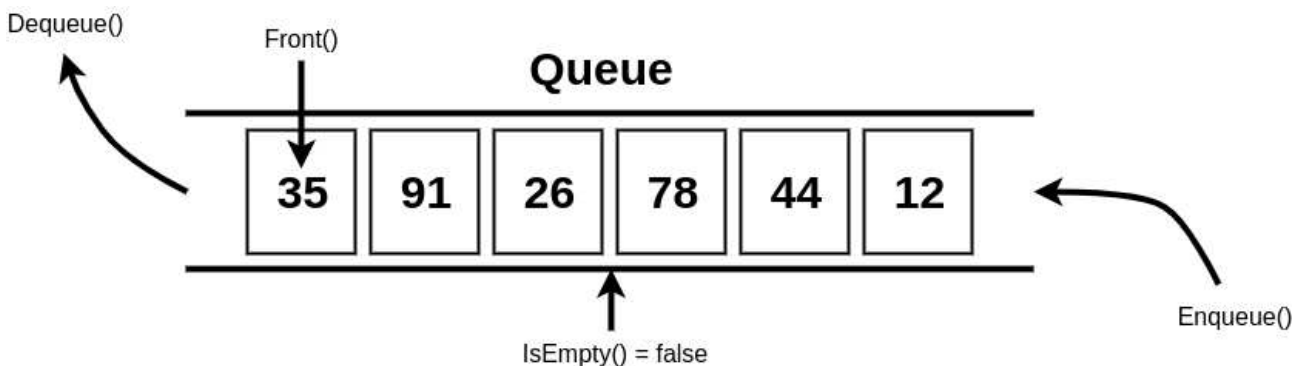
Queue ADT를 C++를 이용하여 구현하세요.

또한 구현이 제대로 되었는지 확인할 수 있는 C++버전의 client 함수(테스트 함수)도 같이 구현하여 테스트하기 바랍니다.

이미 8-queens problem 과제를 위하여 C 버전의 Queue ADT를 구현했기 때문에 쉽게 할수 있는 숙제입니다.

단, Generic coding을 위해 Template 기능은 사용하지 않아도 무관함.

2, 이론 (과제와 관련한 일반적인 내용)



3. 알고리즘 및 자료구조 설계 내용 기술

Queue ADT

- Like a stack, a *queue* is also a list. However, with a queue, insertion is done at one end, while deletion is performed at the other end.
- Accessing the elements of queues follows a **First In, First Out (FIFO)** order.
 - Like customers standing in a check-out line in a store, the first customer in is the first customer served.



4. 소스코드 설명 (직접 작성한 소스코드중에 핵심 부분을 발췌하여 설명)

```
class Queue {
private:
    int *data; // queue
    int front;
    int rear;
public:
    Queue(int size); //생성자
    ~Queue(); //소멸자
    bool is_empty(); //queue가 비어있는 확인하는 함수
    bool is_full(); //queue가 꽉 차있는 확인하는 함수
    void Enqueue(int x); //queue에 data 삽입
    int Dequeue(); //FIFO 순으로 data 삭제
    int QueueFront(); //앞에 있는 data를 검색
    int QueueRear(); //뒤에 있는 data를 검색
    void print(); // 출력하는 함수
    void test(); //테스트 함수
};
```

class Queue를 만들어, private 부분에 data와 front, rear를 만들었다. 그리고 생성자를 통해 동적 메모리를 할당 받고, 소멸자를 통해 동적 메모리를 반납했다. public 부분에는 여러 함수들을 선언했다.

```

void Queue::test() {
    for (int i = 1; i < 20; i += 2) {
        Enqueue(i);
    }    // 1,3,5,7,9,11,13,15,17,19 를 queue에 넣는다.
    print(); //출력
    for (int i = 0; i < 7; i++) {
        Dequeue();
    }    // 처음에 들어간 수부터 7개를 뺀다.
    print(); //출력
    for (int i = 25; i > 15; i--) {
        Enqueue(i);
    }    // 25,24,23,22,21,20,19,18,17,16 을 queue에 넣는다.
    print(); //출력
    for (int i = 0; i < 5; i++) {
        Dequeue();
    }    // 처음에 들어간 수부터 5개를 뺀다.
    print(); //출력
}

void Queue::test() {
    for (int i = 2; i < 20; i += 2) {
        Enqueue(i);
    }    // 2,4,6,8,10,12,14,16,18 을 queue에 넣는다.
    print(); //출력
    for (int i = 0; i < 7; i++) {
        Dequeue();
    }    // 처음에 들어간 수부터 7개를 뺀다.
    print(); //출력
    for (int i = 7; i > 2; i--) {
        Enqueue(i);
    }    // 7,6,5,4,3 을 queue에 넣는다.
    print(); //출력
    for (int i = 0; i < 5; i++) {
        Dequeue();
    }    // 처음에 들어간 수부터 5개를 뺀다.
    print(); //출력
}

```

구현이 제대로 되었는지 확인할 수 있는 Queue클래스의 test함수이다.

위의 함수는 코드1의 test함수이고, 아래의 함수는 코드2의 test함수이다.

코드1의 test함수에서는 처음에 1,3,5,7,9,11,13,15,17,19를 queue에 넣고, 처음에 들어간 수부터 7개를 빼준다. 그리고 25,24,23,22,21,20,19,18,17,16을 queue에 넣고, 처음에 들어간 수부터 5개를 뺀다. 코드2의 test함수에서는 처음에 2,4,6,8,10,12,14,16,18을 queue에 넣고, 처음에 들어간 수부터 7개를 빼준다. 그리고 7,6,5,4,3을 queue에 넣고, 처음에 들어간 수부터 5개를 뺀다.

```

void Queue::Enqueue(int x) {
    if (is_full()) {
        cout << "큐가 포화상태 입니다." << '\n';
        return;
    }
    rear = (rear + 1) % MAX_SIZE; //원형큐이다.
    data[rear] = x;
}

int Queue::Dequeue() {
    if (is_empty()) {
        cout << "큐가 비어있습니다." << '\n';
        return 0;
    }
    front = (front + 1) % MAX_SIZE; //원형큐이다.
    return data[front];
}

```

원형큐이기 때문에 `rear = (rear + 1) % MAX_SIZE`을 하고 `data[rear] = x`를 통해 queue에 x값을 대입해준다. 또한 `front = (front + 1) % MAX_SIZE`를 하고 `return data[front]`를 통해 queue에서 값을 삭제해주고 리턴해준다.

```

void Queue::Enqueue(int data) {
    QueueNode *temp = (QueueNode *)malloc(sizeof(QueueNode)); //동적 메모리를 할당
    받아서 data 삽입
    temp->data = data;
    temp->link = NULL;
    if (is_empty()) {
        front = temp;
        rear = temp;
    }
    else {
        rear->link = temp; //link를 통해 queue 연결
        rear = temp; //rear가 마지막 data 노드를 가리킴.
    }
}

int Queue::Dequeue() {
    QueueNode *temp = front; //temp가 front가 가리키던 노드를 가리키게 함.
    int data;
    if (is_empty()) {
        cout << "큐가 비어있습니다.";
        return 0;
    }
    else {
        data = temp->data;
        front = front->link;
        if (front == NULL) {
            rear = NULL;
        }
        free(temp); //동적 메모리 반납
        return data;
    }
}

```

QueueNode를 가리키는 temp를 하나 만들고 동적 메모리를 할당 받는다. temp가 가리키는 노드의 data를 parameter data로 대입하고, link에는 NULL 값을 넣어준다. Queue가 비어있다면 front와 rear가 모두 새로 만들어진 노드를 가리키게 하고, Queue에 값이 있었다면, rear가 가리키던 노드의 link가 새로 만들어진 노드를 가리키게 하고, rear는 새로 만들어진 노드를 가리킨다. 삭제 할 때도 마찬가지로 QueueNode를 가리키는 temp를 하나 만들고 front가 가리켰던 노드에 대입한다. temp가 가리키는 노드의 data를 따로 저장해주고, front가 가리키던 노드의 link가 가리키는 노드를 front가 가리키게 한다. 그리고 temp를 동적 메모리 반납하여 data를 삭제해준다. 아까 미리 저장해두었던 data 값을 리턴한다.

```

void Queue::print() {
    QueueNode *p;
    cout << "Queue : ";
    for (p = front; p != NULL; p = p->link) { //반복문을 통해 queue 출력
        cout << p->data << " ";
    }
    cout << "\n";
}

```

QueueNode를 가리키는 변수 p를 만들고, front가 가리키는 노드를 p가 가리키게 한다. 그리고 p가 가리키던 노드의 link가 가리키던 노드를 p가 가리키게 하여 queue의 값들을 하나씩 출력한다.

5. 실행결과 및 설명 (실행 결과를 캡처하여 첨부한 후 설명)

 Microsoft Visual Studio 디버그 콘솔


```

Queue : 1 3 5 7 9 11 13 15 17 19
Queue : 15 17 19
Queue : 15 17 19 25 24 23 22 21 20 19 18 17 16
Queue : 23 22 21 20 19 18 17 16

```

C:\Users\seomk\source\repos\HW6하상천\Debug\HW6하상천.exe(634
 디버깅이 중지될 때 콘솔을 자동으로 닫으려면 [도구]->[옵션]->[디버깅]
 섹션 설정합니다.
 이 창을 닫으려면 아무 키나 누르세요.

코드 1을 실행한 결과화면입니다.

 Microsoft Visual Studio 디버그 콘솔

```

Queue : 2 4 6 8 10 12 14 16 18
Queue : 16 18
Queue : 16 18 7 6 5 4 3
Queue : 4 3

```

C:\Users\seomk\source\repos\HW6하상천\Debug\HW6하상천.exe(8164 프
 디버깅이 중지될 때 콘솔을 자동으로 닫으려면 [도구]->[옵션]->[디버깅]
 섹션 설정합니다.
 이 창을 닫으려면 아무 키나 누르세요.

코드2를 실행한 결과화면입니다.

(그림을 문서에 포함, 글자처럼 취급 옵션, 잉크 절약과 잘 보이게 하기위해 그림 반전)

6. 고찰 (과제를 진행하면서 배운점 이나, 시행 착오 내용, 기타 느낀점)

이번 과제는 Queue ADT를 c++을 이용하여 구현해보는 것이었다. c++을 공부한지 얼마 안돼서 처음에는 출력문과 입력문, class 등등 어색한 것이 많이 있었다. 하지만 많은 시간을 투자해서 코딩해보고, 공부해보니 많이 익숙해졌다. Queue ADT는 예전에 공부해본적이 있어서 쉽게 구현할 수 있었다. 두 가지 방법으로 구현해보았는데 첫 번째 방법은 배열을 이용하여 원형큐로 만드는 것이고, 두 번째 방법은 저번 과제 때 했던 Linked list를 이용하여 queue를 구현하는 것이다. 첫 번째 방법은 20 size 만큼 동적 메모리를 할당 받고, Enqueue, Dequeue 함수를 통해 배열에 data를 삽입하고, 삭제하였다. 원형큐이기 때문에 삽입할 때 $rear = (rear + 1) \% MAX_SIZE$ 를 통해 data를 삽입하였고, $front = (front + 1) \% MAX_SIZE$ 를 통해 data를 삭제하였다. 왜냐하면 MAX_SIZE로 나눈 나머지로 계산해야 배열의 인덱스를 초과했을 때 다시 배열의 인덱스 값으로 바꾸어줄 수 있기 때문이다. 두 번째 방법은 data를 삽입 할 때마다 data node를 하나씩 만들어서 data 값을 대입해주고, link로 queue를 연결해 주었다. data를 삭제 할 때는 data node를 가리키는 임의의 변수를 하나 만들어서 front가 가리키는 노드를 가리키게 하고, front는 front가 가리키던 노드의 link가 가리키는 노드를 가리킨다. 그리고 임의의 변수가 가리키는 data를 따로 저장해주고, 동적 메모리 반납을 통해 data node를 삭제한다. 처음에는 c++이 엄청 어렵게 느껴지고, 자바가 있는데 왜 c++까지 객체지향언어를 해야 하는지 의문점이 들었지만 과제를 통해서 여러 방면으로 생각해보고, 여러 가지 방법으로 코드를 구성해보니 자바와 약간은 다르다는 것을 느꼈고 이전보다는 조금 더 c++에 거부감이 없어진 것 같다.

7. 전체 소스코드 (글자크기 9에 줄간격을 120%로 유지하고 한 줄이 너무 길지 않게 작성)

<코드1>

```
#include <iostream>
using namespace std;
const int MAX_SIZE = 20; // 최대 크기 상수 20

class Queue {
private:
    int *data; // queue
    int front;
    int rear;
public:
    Queue(int size); //생성자
    ~Queue(); //소멸자
    bool is_empty(); //queue가 비어있는 확인하는 함수
    bool is_full(); //queue가 꽉 차있는 확인하는 함수
    void Enqueue(int x); //queue에 data 삽입
    int Dequeue(); //FIFO 순으로 data 삭제
    int QueueFront(); //앞에 있는 data를 검색
    int QueueRear(); //뒤에 있는 data를 검색
    void print(); // 출력하는 함수
```

```

        void test(); //테스트 함수
};

Queue::Queue(int size) {
    data = (int*)malloc(size * sizeof(int)); //size만큼 동적 메모리 할당
    if (data == NULL) {
        free(data);
    }
    front = -1; //front -1로 초기화
    rear = -1; //rear -1로 초기화
}

Queue::~Queue() {
    free(data); //동적 메모리 반납
}

bool Queue::is_empty() {
    return front == rear;
}

bool Queue::is_full() {
    return ((rear+1)% MAX_SIZE) == front; //원형큐이다.
}

void Queue::Enqueue(int x) {
    if (is_full()) {
        cout << "큐가 포화상태 입니다." << 'Wn';
        return;
    }
    rear = (rear + 1) % MAX_SIZE; //원형큐이다.
    data[rear] = x;
}

int Queue::Dequeue() {
    if (is_empty()) {
        cout << "큐가 비어있습니다." << 'Wn';
        return 0;
    }
    front = (front + 1) % MAX_SIZE; //원형큐이다.
    return data[front];
}

int Queue::QueueFront() {
    return data[(front + 1) % MAX_SIZE];
}

```



```

int Queue::QueueRear() {
    return data[rear % MAX_SIZE];
}

void Queue::print() {
    cout << "Queue : ";
    if (!is_empty()) {
        int num = front;
        do {
            num = (num + 1) % (MAX_SIZE);
            cout << data[num] << " ";
            if (num == rear) { //모두 다 출력 했다는 의미이므로 break문을 통해 반
복문 탈출
                                break;
                            }
        } while (num != front);
    }
    cout << "\n";
}

void Queue::test() {
    for (int i = 1; i < 20; i += 2) {
        Enqueue(i);
    }    // 1,3,5,7,9,11,13,15,17,19 를 queue에 넣는다.

    print(); //출력

    for (int i = 0; i < 7; i++) {
        Dequeue();
    }    // 처음에 들어간 수부터 7개를 뺀다.

    print(); //출력

    for (int i = 25; i > 15; i--) {
        Enqueue(i);
    }    // 25,24,23,22,21,20,19,18,17,16 을 queue에 넣는다.

    print(); //출력

    for (int i = 0; i < 5; i++) {
        Dequeue();
    }    // 처음에 들어간 수부터 5개를 뺀다.

    print(); //출력

```

```

}

int main() {
    Queue q(20);
    q.test();

    return 0;
}

```

<코드2>

```

#include <iostream>
using namespace std;

typedef struct QueueNode { //Data node 구조체
    int data;
    struct QueueNode *link;
}QueueNode;

class Queue {
private:
    QueueNode *front;
    QueueNode *rear;
public:
    Queue(); //생성자
    bool is_empty(); //queue가 비어있는 확인하는 함수
    bool is_full(); //queue가 꽉 차있는 확인하는 함수
    void Enqueue(int data); //queue에 data 삽입
    int Dequeue(); //FIFO 순으로 data 삭제
    void print(); // 출력하는 함수
    void test(); //테스트 함수
};

Queue::Queue() {
    front = 0;
    rear = 0;
}

```

```

bool Queue::is_empty() {
    return front == NULL;
}

bool Queue::is_full() {
    return 0;
}

void Queue::Enqueue(int data) {
    QueueNode *temp = (QueueNode *)malloc(sizeof(QueueNode)); //동적 메모리를 할당
    받아서 data 삽입
    temp->data = data;
    temp->link = NULL;
    if (is_empty()) {
        front = temp;
        rear = temp;
    }
    else {
        rear->link = temp; //link를 통해 queue 연결
        rear = temp; //rear가 마지막 data 노드를 가리킴.
    }
}

int Queue::Dequeue() {
    QueueNode *temp = front; //temp가 front가 가리키던 노드를 가리키게 함.
    int data;
    if (is_empty()) {
        cout << "큐가 비어있습니다.";
        return 0;
    }
    else {
        data = temp->data;
        front = front->link;
        if (front == NULL) {
            rear = NULL;
        }
        free(temp); //동적 메모리 반납
        return data;
    }
}

void Queue::print() {
    QueueNode *p;
    cout << "Queue : ";
    for (p = front; p != NULL; p = p->link) { //반복문을 통해 queue 출력

```

```

        cout << p->data << " ";
    }
    cout << "\n";
}

void Queue::test() {
    for (int i = 2; i < 20; i += 2) {
        Enqueue(i);
    }    // 2,4,6,8,10,12,14,16,18 을 queue에 넣는다.

    print(); //출력

    for (int i = 0; i < 7; i++) {
        Dequeue();
    }    // 처음에 들어간 수부터 7개를 뺀다.

    print(); //출력

    for (int i = 7; i > 2; i--) {
        Enqueue(i);
    }    // 7,6,5,4,3 을 queue에 넣는다.

    print(); //출력

    for (int i = 0; i < 5; i++) {
        Dequeue();
    }    // 처음에 들어간 수부터 5개를 뺀다.

    print(); //출력
}

int main() {
    Queue q;
    q.test(); //테스트 함수

    return 0;
}

```

(글자크기는 10으로 유지하고 줄간격도 160%를 유지할 것)