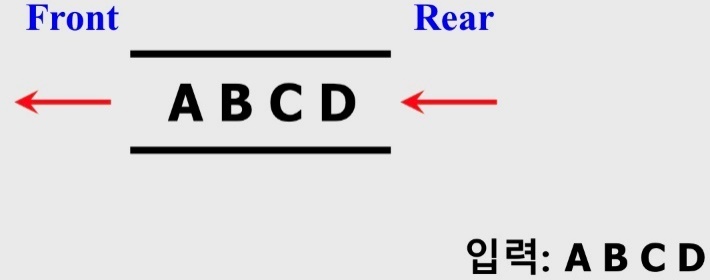
**06. 큐(Queue)**

**06-1. 큐의 이해**

: 리스트의 한쪽 끝에서 삽입 작업이 이루어지고 반대쪽 끝에서 삭제 작업이 이루어져서 삽입된 순서대로 삭제되는 구조

* **선입선출(FIFO, First-In First-Out)**



**06-2. 큐 응용**

**# 다양한 큐 활용**

* **회문(Paindromes)**
  + 앞에서 읽거나 뒤에서 읽으나 똑같은 단어나 문장

ex) 똑바로 읽어도 거꾸로 읽어도 같은 단어 : 기러기, 토마토, 우영우 등

* **운영체제의 작업 큐**
  + 프로세스 스케줄링 큐(Process Scheduling Queue)
  + 프린터 버터 큐(Printer Buffer Queue)
* **시뮬레이션에서의 큐잉 시스템**
  + 모의실험
  + 큐잉 이론(Queueing Theory)
  + 이벤트 발생시키기(시간 구동 시뮬레이션, 사건 구동 시뮬레이션)

**# queue 클래스(FIFO)**

//C++STL : <queue>

#include <queue>

using namespace std;

queue<DataType> queueName //빈 큐 생성

void push(const value\_type& val); //큐에 데이터 추가

void pop; //큐에 데이터 삭제

value\_type& front(); //큐의 첫 번째 원소 반환

value\_type& back(); //큐의 마지막 원소 반환

bool empty() const; //큐가 비어 있는지 여부 확인

size\_type size()const; //큐의 크기 반환

**# 큐 활용 -- C++ STL(queue class)**

#include <iostream>

#include <queue>

using namespace std;

int main(void) {

queue<int> Q;

Q.push(1);

Q.push(2);

Q.push(3);

cout << "queue is empty : " << Q.empty() << endl;

cout << "queue size : " << Q.size() << endl;

while (!Q.empty()) {

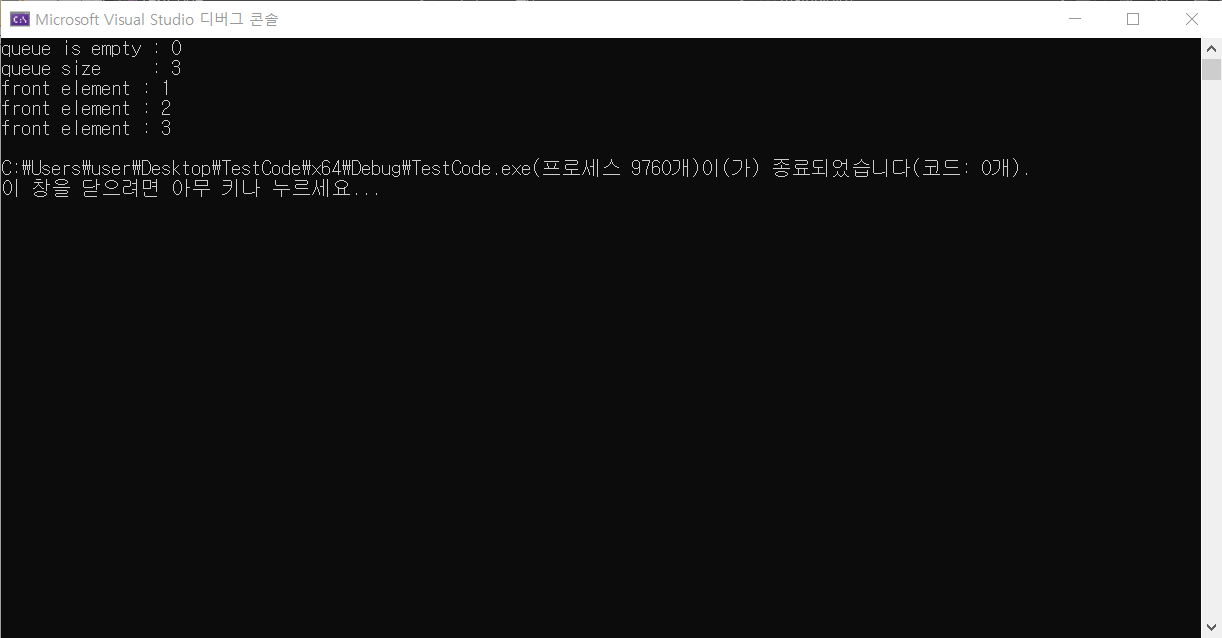
cout << "front element : " << Q.front() << endl;

Q.pop();

}

return 0;

}



**06-3. 큐 구현**

**# 알고리즘**

**1. 초기 공백(선형) 큐 생성**

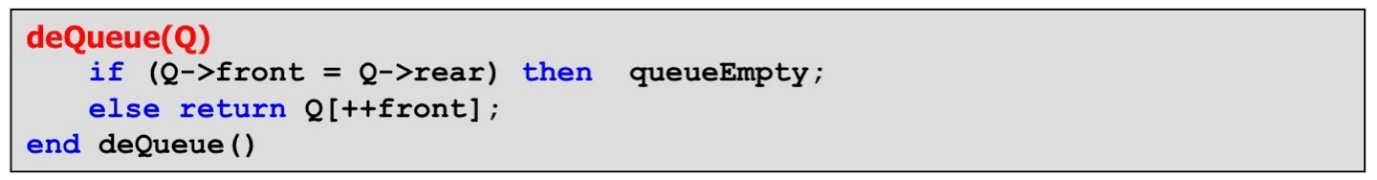


**2. (선형) 큐에 데이터 삽입**

텍스트, 폰트, 라인, 스크린샷이(가) 표시된 사진

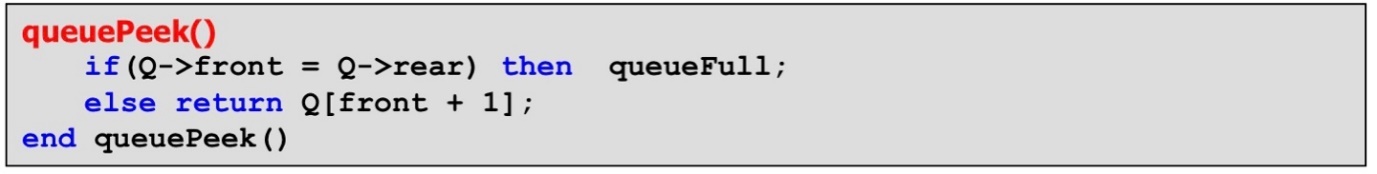
자동 생성된 설명

**3. (선형) 큐에서 데이터 삭제**



**4. (선형) 큐에서 데이터 확인**

: 첫번째로 삽입된 데이터



**5. (선형) 큐의 공백 상태 판단**

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

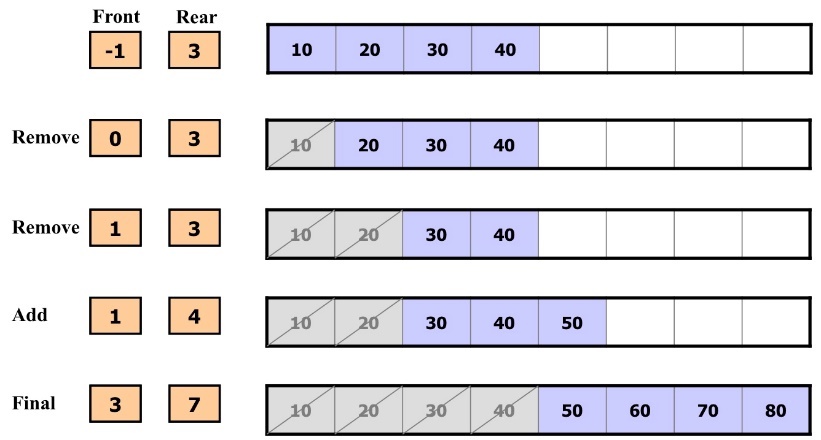
**6. (선형) 큐의 포화 상태 판단**

텍스트, 폰트, 라인, 스크린샷이(가) 표시된 사진

자동 생성된 설명

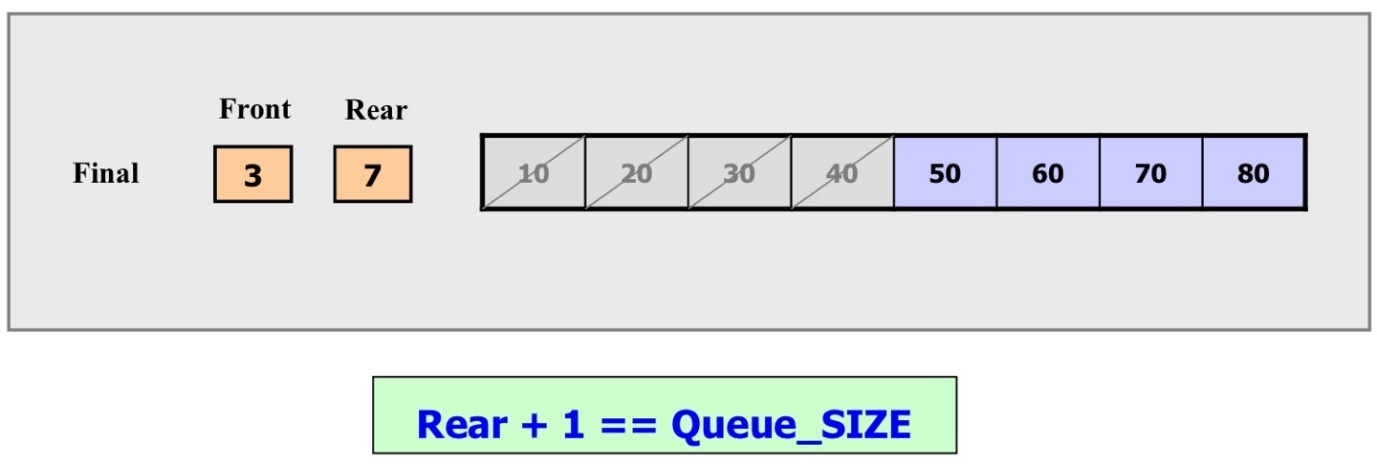
**# 선형 큐(Linear Queue)**

: 연속된 삽입, 삭제에 의한 오른쪽 이동



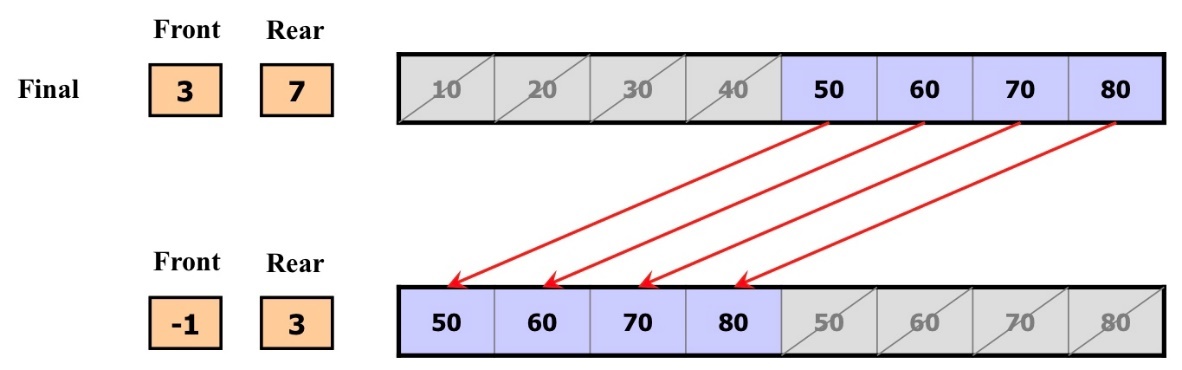
* **선형 큐의 잘못된 포화 상태 인식**

: 큐에서 삽입과 삭제를 반복하면서 아래와 같은 상태일 경우, 앞부분에 빈자리가 있지만 **rear=n-1 상태이므로 포화 상태로 인식하고** 더 이상의 삽입을 수행하지 않는다.

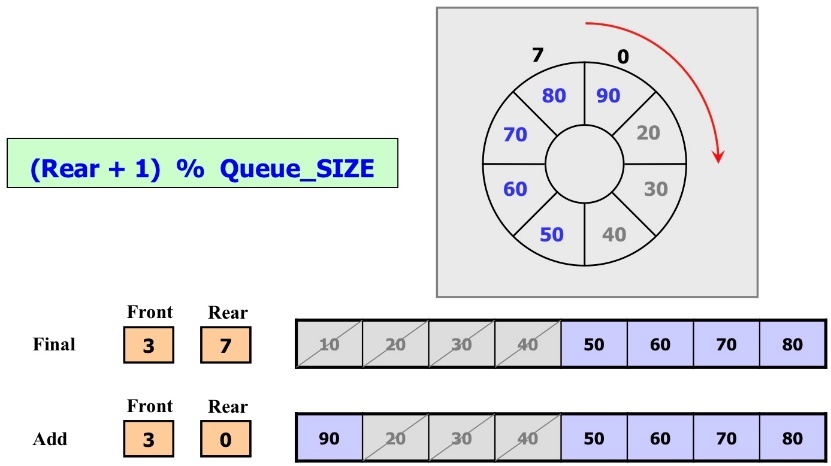


* **해결 방법#1**

: 저장된 원소들을 배열의 앞부분으로 이동(순차 자료에서의 이동 작업은 연산이 복잡하여 효율성이 떨어짐)



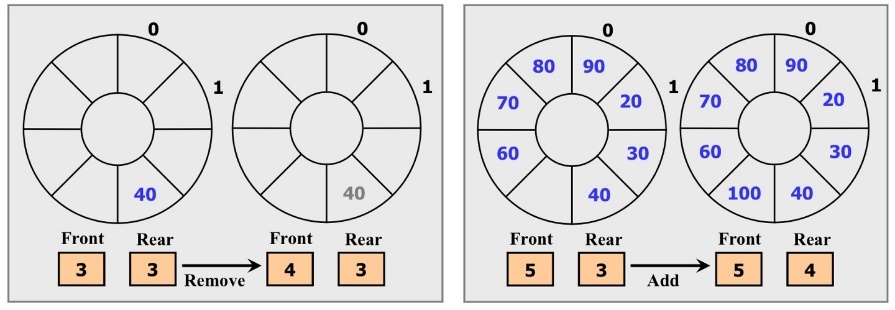
* **원형 큐(선형 큐의 잘못된 포화상태 인식의 해결방법 #2)**

****

* **문제점**

: 빈 큐와 꽉 찬 큐의 판정불가(Front = Rear+1)

* **별도의 Count 변수 유지**

****

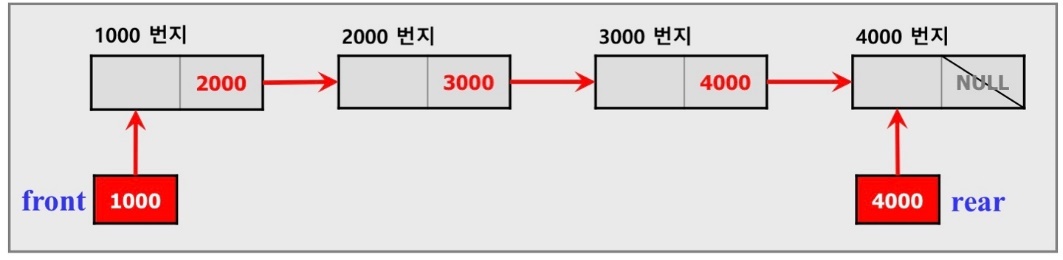
**# 연결 큐(Linked Queue)**

* **순차 자료구조로 구현한 큐의 문제점**
  + 크기가 제한되어 큐의 길이를 마음대로 사용할 수 없다.
  + 원소가 없을 때에도 항상 그 크기를 유지하고 있어야 하므로 메모리 낭비
* **연결 큐의 구조**
  + 데이터 필드와 링크 필드를 가진 노드로 구성

**- front** : 첫 번째 노드를 가리키는 포인터

**- rear** : 마지막 노드를 가리키는 포인터

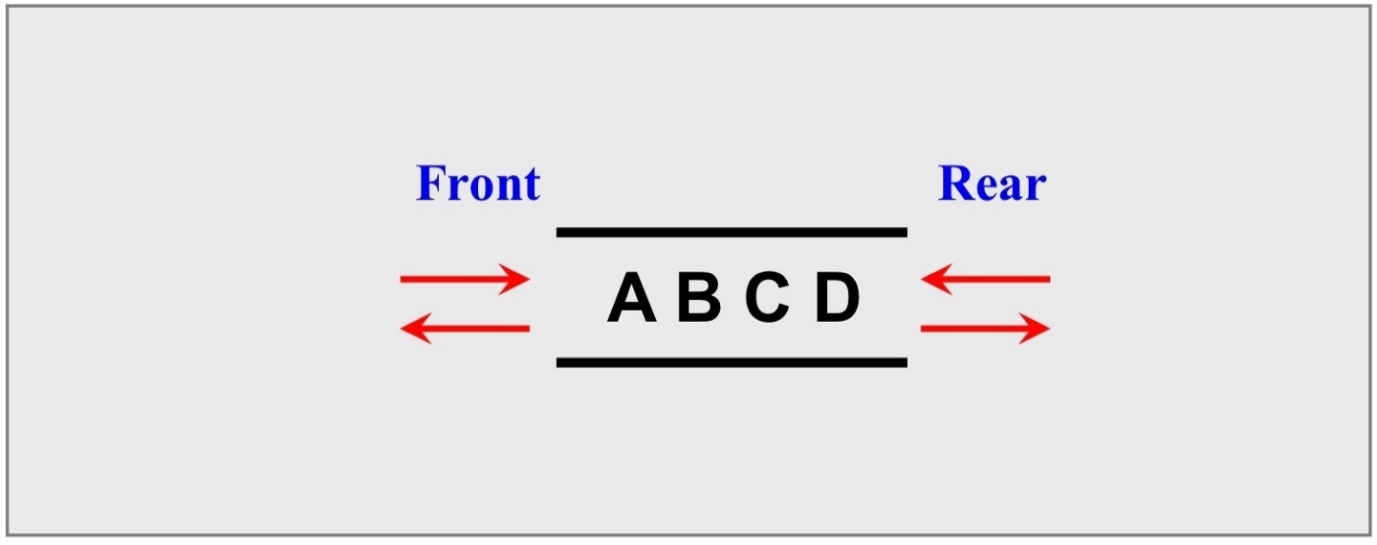
**- 초기 상태(공백 큐**) : front와 rear 모두 널(NULL) 포인터로 설정

****

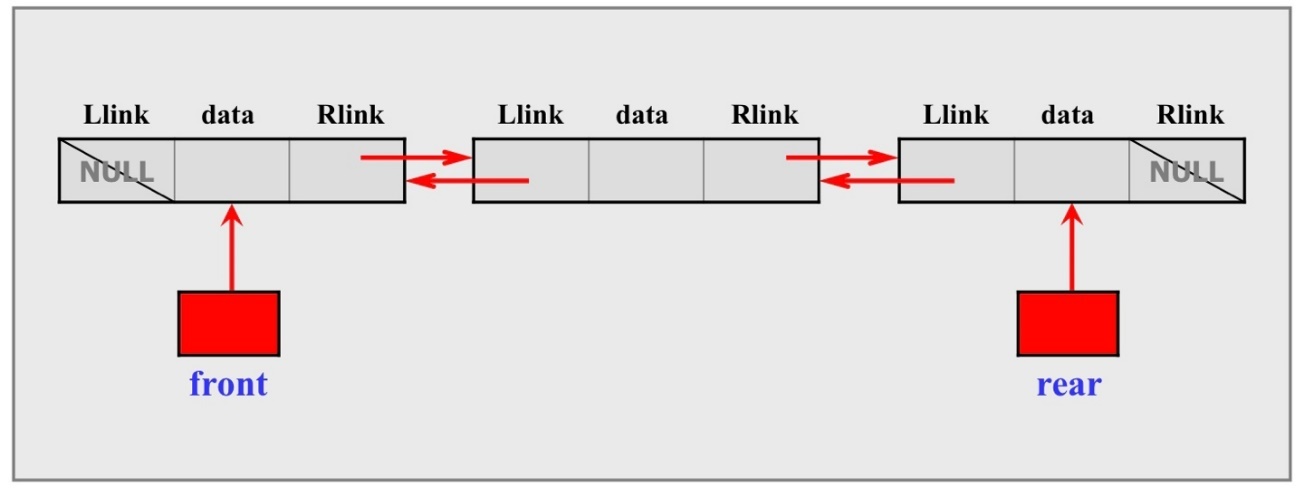
**# 덱(Deque, Double-ended Queue)**

: 큐의 양쪽 끝에서 삽입과 삭제가 모두 발생할 수 있는 큐

* **스택과 큐의 성질을 모두 가지고 있는 자료구조**

****

* **이중 연결리스트 구조를 이용한 덱의 구현**

****

**# 큐 구현 : 순차 자료구조**

**1. ver.C**

#define queueMAXSIZE 100;

typedef int element;

typedef struct \_arrayQueue {

element queue[queueMAXSIZE];

int front, rear;

}arrayQueue;

arrayQueue\* queueCreate(void);

void queueDestroy(arrayQueue\* Q);

void enQueue(arrayQueue\* Q, element item);

element deQueue(arrayQueue\* Q);

element peek(arrayQueue\* Q);

int isEmpty(arrayQueue\* Q);

int isFull(arrayQueue\* Q);

void queuePrint(arrayQueue\* Q);

**2. ver.CPP**

#define queueMAXSIZE 100;

template<typename E>

class arrayQueue {

private:

E queue[queueMAXSIZE];

int front, rear;

public:

arrayQueue();

~arrayQueue();

void enQueue(const E& e);

E deQuee(void);

E peek(void) const;

bool isEmpty(void) const;

bool isFull(void) const;

void printQueue(void) const;

};

**# 큐 구현 : 연결 자료구조**

**1. LinkedQueue#1(ver.C)**

**# LinkedNode.c**

/\*

단순.이중 연결 리스트 : 알고리즘 구현(C)

파일명: LinkedNode.c

- 노드 : SNode(data, link)

- 노드 : DNode(data, link)

\*/

#include <stdio.h>

#include <stdlib.h> // malloc, free

#include "LinkedNode.h" // SNode, DNode

// 단순 연결 리스트 구현(C)

// 새로운 노드(data, link) 생성

SNode\* makeSNode(element num) {

SNode\* newSNode = (SNode\*)malloc(sizeof(SNode));

if (newSNode == NULL) {

printf("노드 생성 실패!!! \n");

exit(100);

}

newSNode->\_\_data = num;

newSNode->\_\_link = NULL;

return newSNode;

}

// 이중 연결 리스트 구현(C)

// 새로운 노드(DNode: data, Llink, Rlink) 생성

DNode\* makeDNode(element num) {

DNode\* newDNode = (DNode\*)malloc(sizeof(DNode));

if (newDNode == NULL) {

printf("노드 생성 실패!!! \n");

exit(100);

}

newDNode->\_\_data = num;

newDNode->\_\_Llink = NULL;

newDNode->\_\_Rlink = NULL;

return newDNode;

}

**# LinkedQueue(demo).c**

/\*

큐: 알고리즘 구현 -- 단순연결리스트

- main : 데이터 삽입.삭제, 전체 원소 출력

\*/

#include <stdio.h>

#include <stdlib.h> // system

#include <stdbool.h> // bool, true, false

#include "LinkedQueue.h" // LinkedQueue

int main(void)

{

int num, data;

LinkedQueue\* Q = queueCreate();

while (true) {

system("cls");

printf("\n ### 큐 구현: 단순 연결 리스트 ### \n\n");

printf("1) 데이터 삽입(enQueue) \n");

printf("2) 데이터 삭제(deQueue) \n");

printf("3) 전체 출력 \n");

printf("4) 프로그램 종료 \n\n");

printf("메뉴 선택: ");

scanf\_s("%d", &num);// scanf("%d", &num);

switch (num) {

case 1: printf("\n삽입 할 데이터 입력 : ");

scanf\_s("%d", &data); // scanf("%d", &data);

enQueue(Q, data);

break;

case 2: printf("삭제 된 데이터 : %3d \n", front(Q));

deQueue(Q);

break;

case 3:printQueue(Q);

break;

case 4: printf("프로그램 종료... \n");

return 0;

default: printf("잘못 선택 하셨습니다. \n");

}

system("pause");

}

queueDestroy(Q);

return 0;

}

**# LinkedQueue.c**

/\*

큐: 알고리즘 구현 -- 연결자료구조(단순연결리스트)

파일명: LinkedQueue.c

- 큐의 생성.소멸 : queueCreate, queueDestroy

- 데이터 삽입.삭제 : enQueue, deQueue

- 데이터 확인(peek) : front, back

- 빈 스택 여부 판단 : queueEempty

- 큐의 크기 : queueSize

- 큐의 전체 원소 출력 : printQueue

\*/

#include <stdio.h>

#include <stdlib.h> // malloc, free

#include "LinkedQueue.h" // LinkedQueue, SNode

// queueCreate : 빈 큐 생성

LinkedQueue\* queueCreate(void) {

LinkedQueue\* Q = (LinkedQueue\*)malloc(sizeof(LinkedQueue));

if (Q == NULL) {

printf("스택 생성 실패!!! \n");

return NULL;

}

Q->\_\_front = NULL;

Q->\_\_rear = NULL;

return Q;

}

// queueDestroy : 큐 삭제 -- 모든 노드 삭제

void queueDestroy(LinkedQueue\* Q) {

SNode\* temp = Q->\_\_front;

while (temp) {

Q->\_\_front = temp->\_\_link;

free(temp);

temp = Q->\_\_front;

}

free(Q);

return;

}

// enQueue : 큐에 데이터 삽입

void enQueue(LinkedQueue\* Q, element data) {

SNode\* newSNode = makeSNode(data);

if (Q->\_\_front == NULL) {

Q->\_\_front = newSNode;

Q->\_\_rear = newSNode;

}

else {

Q->\_\_rear->\_\_link = newSNode;

Q->\_\_rear = newSNode;

}

}

// deQueue : 큐에서 데이터 삭제

void deQueue(LinkedQueue\* Q) {

if (queueEempty(Q)) return;

SNode\* temp = Q->\_\_front;

Q->\_\_front = temp->\_\_link;

if (Q->\_\_front == NULL)

Q->\_\_rear = NULL;

free(temp);

}

// front : 큐에서 첫 번째 원소 확인

element front(LinkedQueue\* Q) {

if (queueEempty(Q)) return EOF; // return NULL;

return Q->\_\_front->\_\_data;

}

// back : 큐에서 맨 마지막 원소 확인

element back(LinkedQueue\* Q) {

if (queueEempty(Q)) return EOF; // return NULL;

return Q->\_\_rear->\_\_data;

}

// queueEempty : 큐의 공백 상태 여부 판단

\_Bool queueEempty(LinkedQueue\* Q) {

return Q->\_\_front == NULL;

}

// queueSize: 큐의 크기

int queueSize(LinkedQueue\* Q) {

return Q->\_\_count;

}

// printQueue : 큐의 전체 원소 출력

void printQueue(LinkedQueue\* Q) {

SNode\* temp = Q->\_\_front;

printf("\n Queue [");

while (temp) {

printf("%3d", temp->\_\_data);

temp = temp->\_\_link;

}

printf(" ]\n");

}

**# LinkedNode.h**

/\*

단순.이중 연결 리스트 : 알고리즘 구현(C)

파일명: LinkedNode.h

- 노드 : SNode(\_\_data, \_\_link)

- 노드 : DNode(\_\_data, \_\_Llink, \_\_Rlink)

\*/

#pragma once

typedef int element;

// 단순 연결 리스트 구현(C)

// 노드: SNode(data, link)

#ifndef \_\_SNode\_H\_\_

#define \_\_SNode\_H\_\_

typedef struct \_\_SNode {

element \_\_data;

struct \_\_SNode\* \_\_link;

}SNode;

#endif

SNode\* makeSNode(element data);

// 이중 연결 리스트 구현(C)

// 노드: DNode(data, Llink, Rlink)

#ifndef \_\_DNode\_H\_\_

#define \_\_DNode\_H\_\_

typedef struct \_\_DNode {

element \_\_data;

struct \_\_DNode\* \_\_Llink;

struct \_\_DNode\* \_\_Rlink;

}DNode;

#endif

DNode\* makeDNode(element data);

**# LinkedQueue.h**

/\*

큐: 알고리즘 구현(C) -- 단순 연결 리스트

파일명: LinkedQueue.h

- 큐의 생성.소멸 : queueCreate, queueDestroy

- 데이터 삽입.삭제 : enQueue, deQueue

- 데이터 확인(peek) : front, back

- 빈 스택 여부 판단 : queueEempty

- 큐의 크기 : queueSize

- 큐의 전체 원소 출력 : printQueue

\*/

#pragma once

#include "LinkedNode.h" // SNode, DNode

// 큐 생성: LinkedQueue

#ifndef \_\_LinkdedQueue\_H\_\_

#define \_\_LinkdedQueue\_H\_\_

typedef struct \_\_LinkedQueue {

SNode\* \_\_front, \* \_\_rear;

int \_\_count;

}LinkedQueue;

#endif

// 큐 구현(C):: 큐 생성 및 활용

LinkedQueue\* queueCreate(void);

void queueDestroy(LinkedQueue\* Q);

void enQueue(LinkedQueue\* Q, element data);

void deQueue(LinkedQueue\* Q);

element front(LinkedQueue\* Q);

element back(LinkedQueue\* Q);

\_Bool queueEempty(LinkedQueue\* Q);

int queueSize(LinkedQueue\* Q);

void printQueue(LinkedQueue\* Q);

**2. LinkedQueue#2(ver.CPP)**

**# LinkedQueue.cpp**

/\*

큐: 알고리즘 구현 -- 단순연결리스트

- main : 데이터 삽입(push, enQueue).삭제(pop, deQueue), 전체 원소 출력

\*/

#include <iostream>

#include "LinkedQueue.h"

using namespace std;

int main(void)

{

int num, choice;

LinkedQueue<int> Q; // 빈 큐 생성

while (true) {

system("cls");

cout << "### 큐 구현: 단순 연결 리스트 ###\n" << endl;

cout << "1) 데이터 삽입(push, enQueue)" << endl;

cout << "2) 데이터 삭제(pop, deQueue)" << endl;

cout << "3) 전체 출력" << endl;

cout << "4) 프로그램 종료 \n" << endl;

cout << "메뉴 선택: ";

cin >> choice;

switch (choice) {

case 1: cout << "\n삽입 할 데이터 입력: ";

cin >> num;

Q.push(num);

break;

case 2: cout << "삭제 된 데이터: " << Q.front() << endl;

Q.pop();

break;

case 3: Q.printQueue();

break;

case 4: cout << "프로그램 종료..." << endl;

return 0;

default: cout << "잘못 선택 하셨습니다." << endl;

}

system("pause");

}

return 0; // Q.~LinkedQueue();

}

**#LinkedQueue.h**

#include <iostream>

using namespace std;

inline void error(const char\* message) {

cout << message;

exit(100);

}

template<typename E>

class SNode {

public:

SNode(const E& data);

E getData(void)const;

private:

E \_\_data;

SNode<E>\* \_\_link;

template<typename E> friend class LinkedQueue;

};

//SNode class: 멤버 함수 외부 정의

template <typename E>

SNode<E>::SNode(const E& data)

: \_\_data(data), \_\_link(nullptr){}

template <typename E>

E SNode<E>::getData(void) const {

return \_\_data;

}

//LinkedQueue class

template <typename E>

class LinkedQueue {

public:

LinkedQueue();

~LinkedQueue();

void push(const E& e);//enQueue

void pop(void); //deQueue

E& front(void) const;

E& back(void) const;

bool empty(void) const;

int size(void)const;

void printQueue(void) const;

private:

SNode<E>\* \_\_front;

SNode<E>\* \_\_rear;

int \_\_count;

};

//LinkedStack class: 생성자(소멸자)와 소멸자

template<typename E>

LinkedQueue<E>::LinkedQueue()

: \_\_front(nullptr), \_\_rear(nullptr), \_\_count(0){}

template <typename E>

LinkedQueue<E> :: ~LinkedQueue() {

SNode<E>\* temp = \_\_front;

while (temp) {

\_\_front = temp->\_\_link;

delete temp;

temp = \_\_front;

}

}

//push(enQueue) : 큐에 데이터 항목 삽입(맨 마지막 원소)

template <typename E>

void LinkedQueue<E>::push(const E& e) {//enQueue

SNode<E>\* newSNode = new SNode<int>(e);

//newSNode-> \_\_data=e;

if (\_\_front)

\_\_rear->\_\_link = newSNode;

else \_\_front = newSNode;

\_\_rear = newSNode;

}

//pop(deQueue) : 큐에서 맨 첫번째로 삽입된 데이터 항목 삭제

template <typename E>

void LinkedQueue<E>::pop(void) {//deQueue

if (empty()) error("STACK IS EMPTY"); //throw "ERROR : STACK IS EMPTY";

SNode<int>\* temp = \_\_front;

//E data = temp->\_\_data;

\_\_front = temp->\_\_link;

if (!\_\_front)

\_\_rear = nullptr;

delete temp;

}

//front(peek) : 큐에서 맨 첫 번째로 삽입된 데이터 항목 확인

template<typename E>

E& LinkedQueue<E>::front(void)const {

if (empty())error("STACK IS EMPTY");//throw "ERROR::STACK IS EMPTY";

return \_\_front->\_\_data;

}

//back(peek) : 큐에서 맨 마지막에 삽입된 데이터 항목 확인

template<typename E>

E& LinkedQueue<E>::back(void)const {

if (empty())error("STACK IS EMPTY"); //throw "ERROR::STACK IS EMPTY";

return \_\_rear->\_\_data;

}

//empty : 스택의 공백 여부 확인

template <typename E>

bool LinkedQueue<E> ::empty(void)const {

return \_\_front == nullptr;

}

//size : 큐의 크기(항목의 개수)

template<typename E>

int LinkedQueue<E> ::size(void)const {

return \_\_count;

}

//printQueue : 큐의 전체 데이터 항목 출력

template <typename E>

void LinkedQueue<E>::printQueue(void)const {

SNode<E>\* temp = \_\_front;

cout << "\nQUEUE [";

while (temp) {

cout << temp->\_\_data;

temp = temp->\_\_link;

}

cout << "]" << endl;

}

**06-4. 원형 큐 구현\_송**

**# 코드**

#include <iostream>

using namespace std;

#define queueMAXSIZE 100

template <typename E>

class arrayQueue {

private:

E queue[queueMAXSIZE];

int front, rear, count;

public:

//생성자 & 소멸자

arrayQueue();

~arrayQueue();

//조사

bool isEmpty(void) const;

bool isFull(void) const;

//출력

void printQueue(void) const;

//enQueue,deQueue,peek

void enQueue(const E& e);

E deQueue(void);

E peek(void) const;

};

template<typename E>

arrayQueue<E>::arrayQueue() {

front = -1;

rear = -1;

count = 0;

}

template<typename E>

arrayQueue<E>::~arrayQueue() {}

template<typename E>

bool arrayQueue<E>::isEmpty() const {

return front == rear;

}

template<typename E>

bool arrayQueue<E>::isFull() const {

return count == queueMAXSIZE;

//return (rear+1)%queueMAXSIZE == front;

}

template<typename E>

void arrayQueue<E>::printQueue() const {

if (isEmpty()) cout << "큐가 비어있습니다. " << endl;

else {

if (rear > front) {

for (int i = front + 1; i <= rear; i++) {

cout.width(3);

cout << queue[i];

}

}

else if (rear < front) {

for (int i = front + 1; i < queueMAXSIZE; i++) {

cout.width(3);

cout << queue[i];

}

for (int i = 0; i <= rear; i++) {

cout.width(3);

cout << queue[i];

}

}

cout << endl;

}

}

template<typename E>

void arrayQueue<E>::enQueue(const E& e) {

if (isFull())

cout << "큐가 꽉 찼습니다." << endl;

else {

rear = (rear + 1) % queueMAXSIZE;

queue[rear] = e;

count++;

cout << "현재 count : " << count << endl;

}

}

template<typename E>

E arrayQueue<E>::deQueue() {

if (isEmpty()) cout << "큐가 비어있습니다. " << endl;

else {

count--;

cout << "현재 count : " << count << endl;

return queue[++front];

}

}

template<typename E>

E arrayQueue<E>::peek() const {

if (isEmpty()) cout << "큐가 비어있습니다. " << endl;

else return queue[front + 1];

}

int main(void)

{

cout << "20215229 이채훈\n" << endl;

int num, choice;

arrayQueue<int> aQ;

while (true) {

system("cls");

cout << "\n ### 큐 구현: 단순 연결 리스트 ### \n" << endl;

cout << "1) 데이터 삽입: enQueue" << endl;

cout << "2) 데이터 삭제: deQueue" << endl;

cout << "3) 전체 출력" << endl;

cout << "4) 프로그램 종료 \n" << endl;

cout << "메뉴 선택 : ";

cin >> choice;

switch (choice) {

case 1: cout << "\n삽입 할 데이터 입력: ";

cin >> num;

aQ.enQueue(num);

break;

case 2: cout << "삭제 된 데이터: " << aQ.deQueue() << endl;

break;

case 3: aQ.printQueue();

break;

case 4: cout << "프로그램 종료..." << endl;

return 0;

default: cout << "잘못 선택 하셨습니다." << endl;

}

system("pause");

}

return 0;

}

**# 연구조사**

큐는 아래와 같이 이해할 수 있다.

생성자&소멸자

생성-makeQueueNode

탐색-isEmpty(), isFull() (X)

출력-printStack

큐함수- enqueue, dequeue, peek

isEmpty

큐가 비어있는지 확인한다. front와 rear가 같으면 큐가 비었다고 판단한다.

isFull

큐가 꽉 찼는지 확인한다. 선형 큐의 경우, rear+1의 값이 queueMaXSIZE와 같은지 확인하면 되나,

원형 큐는 같은 방법으로 할 경우, front = rear인 경우가 생겨, isEmpty와 구분할 수 없다.

따라서 별도의 count 변수를 유지하여 데이터의 갯수를 세고, count == queueMaXSIZE와 같은지 확인한다.

다른 방법으로는, 한칸의 큐를 쓰지 않는 방법이 있는데, 이 때는 (rear+1)%queueMaXSIZE가 front와 같은지 확인하는 방법을 사용한다

printQueue

큐의 모든 데이터를 확인한다.

rear > front인 경우는 선형 큐와 같은 상황이므로 front+1부터 rear까지 출력하면 되나,

rear < front인 경우는 queue의 마지막 칸까지 채워진 상태에서, 비어있는 앞칸에 데이터를 삽입하는 경우이므로 두 부분을 나누어 출력해준다.

enQueue

rear에 data를 입력한다.

(rear+1)%queueMAXSIZE의 의미 : rear값을 반드시 0~7 사이값으로 만든다.

이는 rear가 queueMAXSIZE-1 의 인덱스에 도달할 경우, rear를 다시 0부터 시작하도록 해주며, 앞칸부터 다시 삽입할 수 있도록 해준다

1. 꽉 차있는지 확인

2. 꽉 차지 않은 경우 rear = (rear+1)%queueMAXSIZE로 삽입할 자리 선정

3. 이후 삽입 실행

4. count 갯수 증가

deQueue

front에서 data를 삭제한다.

1. 비어있는지 확인

2. 비어있지 않은 경우, count 갯수 감소 후 , 현재 front의 다음칸을 반환

peek

front에 data를 반환한다.

**06-5. 연결리스트 큐 구현\_송**

**# 코드**

#include <iostream>

using namespace std;

inline void error(const char\* message);

template<typename E>

class QueueNode {

E data;

QueueNode<E>\* link;

template<typename E> friend class LinkedQueue

};

inline void error(const char\* message) {

cout << message;

exit(100); //exit :

}

template<typename E>

class LinkedQueue {

QueueNode<E>\* front, \* rear; //E로 만들어진 front,rear;

public:

//생성자 & 소멸자

LinkedQueue();

~LinkedQueue();

//생성

QueueNode<E>\* makeQueueNode(const E& e);

//조사

bool isEmpty() const;

//출력

void printQueue() const;

//삽입,삭제,peek

void enQueue(const E& e);

E deQueue();

E peek() const;

};

//생성자 & 소멸자

template<typename E>

LinkedQueue<E>::LinkedQueue() {

front = NULL;

rear = NULL;

}

template<typename E>

LinkedQueue<E>::~LinkedQueue() {}

//생성

template<typename E>

QueueNode<E>\* LinkedQueue<E>::makeQueueNode(const E& e) {

QueueNode<E>\* nNode = new QueueNode<E>;

nNode->data = e;

nNode->link = NULL;

return nNode;

}

//조사

template<typename E>

bool LinkedQueue<E>::isEmpty() const {

return front == NULL;

}

//출력

template<typename E>

void LinkedQueue<E>::printQueue() const {

if (isEmpty()) {

cout << " Queue is Empty " << endl;

return;

}

QueueNode<E>\* temp = front; // 첫 노드부터

while (temp) {

cout.width(3);

cout << temp->data;

temp = temp->link;

}//while문의 결과 : NULL

cout << '\n';

}

template<typename E>

void LinkedQueue<E>::enQueue(const E& e) {

QueueNode<E>\* nNode = makeQueueNode(e); //새로운 노드 생성

if (isEmpty()) { //비어있는 리스트라면

front = nNode;

rear = nNode;

}

else {

rear->link = nNode; // n번째 노드가 new 노드를 가리키도록 함

rear = nNode; // rear에 new 노드 저장

}

}

template<typename E>

E LinkedQueue<E>::deQueue() {

if (isEmpty()) error("큐 공백 에러");

QueueNode<E>\* old = front; //기존 front

E data = old->data; // data 살리고

front = old->link; // front 갱신

free(old);

return data;

}

template<typename E>

E LinkedQueue<E>::peek() const {

if (isEmpty()) {

error("큐 공백 에러");

return;

}

else return front->data;

}

int main(void)

{

cout << "20215229 이채훈\n" << endl;

int num, choice;

LinkedQueue<int> s;

while (true) {

system("cls");

cout << "\n ### 큐 구현: 단순 연결 리스트 ### \n" << endl;

cout << "1) 데이터 삽입: enQueue" << endl;

cout << "2) 데이터 삭제: deQueue" << endl;

cout << "3) 전체 출력" << endl;

cout << "4) 프로그램 종료 \n" << endl;

cout << "메뉴 선택 : ";

cin >> choice;

switch (choice) {

case 1: cout << "\n삽입 할 데이터 입력: ";

cin >> num;

s.enQueue(num);

break;

case 2: cout << "삭제 된 데이터: " << s.deQueue() << endl;

break;

case 3: s.printQueue();

break;

case 4: cout << "프로그램 종료..." << endl;

return 0;

default: cout << "잘못 선택 하셨습니다." << endl;

}

system("pause");

}

return 0;

}

**# 연구조사**

선입 선출

스택은 "후입선출"로 항상 top에서 push와 pop을 수행한다.

반면 큐는 "선입선출"으로 rear에서 push하고 front에서 pop을 수행한다.

아래와 같은 구조로 이해할 수 있다.

생성자&소멸자

생성-makeQueueNode

탐색-isEmpty(), //// isFull() (X)

출력-printStack

stack함수 - enqueue, dequeue, ,peek

enqueue

큐에 원소를 삽입한다.

rear->link : 기존의 맨 뒤가 새로운 노드를 가리키도록

rear = nNode : 마지막 노드 갱신

deQueue

큐에서 원소를 삭제한다

1. old에 삭제할 노드(front)를 백업

2. 노드를 삭제한 후에도, data를 return 할 수 있도록 백업

3. 2번째 노드를 front로

4. 노드 삭제

5. 삭제한 노드의 data 리턴

peek

큐에 가장 먼저 들어온 data를 확인

1. 빈 큐인지 확인

2. 비어있지 않으면 front의 데이터 반환

system("cls") : 실행됐던 기존 콘솔 화면을 삭제한다.

system("pause") : 콘솔 화면을 멈추는 함수. 원활하게 결과를 확인하고자 하는 부분에 넣어준다.

switch문의 입력값에는 정수형만 가능하다. 입력값으로 문자열을 넣을 경우, 이해할 수 없는 오류가 발생한다.

**06-6. 3차 실습보고서\_연습문제#03\_원형 큐 구현(순차 자료구조)**

**# 코드**

#include <iostream>

using namespace std;

inline void error(const char\* message);

#define queueMAXSIZE 7

template <typename E>

class CircularQueue {

private:

E queue[queueMAXSIZE];

int \_\_front, \_\_rear, \_\_count;

public:

//생성자 & 소멸자

CircularQueue();

~CircularQueue();

//push, pop,front, back

void push(const E& e);

E pop(void);

E front(void) const;

E back(void) const;

//size, empty, full

int size(void) const;

bool empty(void) const;

bool full(void) const;

//printQueue

void printQueue(void) const;

};

template<typename E>

CircularQueue<E>::CircularQueue()

: \_\_front(-1), \_\_rear(-1), \_\_count(0) {}

template<typename E>

CircularQueue<E>::~CircularQueue() {}

//push(enQueue): 큐에 데이터 항목 삽입(맨 마지막 원소)

template<typename E>

void CircularQueue<E>::push(const E& e) {

if (full())

error("원형 큐 포화 에러");

else {

\_\_rear = (\_\_rear + 1) % queueMAXSIZE;

queue[\_\_rear] = e;

\_\_count++;

}

}

//pop(deQueue): 큐에서 맨 첫 번째로 삽입된 데이터 항목 삭제

template<typename E>

E CircularQueue<E>::pop() {

if (empty())

error("원형 큐 공백 에러");

else {

\_\_count--;

return queue[++\_\_front];

}

}

//front(peek): 큐에서 맨 첫 번째로 삽입된 데이터 항목 확인

template<typename E>

E CircularQueue<E>::front() const {

if (empty())

error("원형 큐 공백 에러");

else

return queue[(\_\_front + 1) % queueMAXSIZE];

}

//back(peek): 큐에서 맨 마지막에 삽입된 데이터 항목 확인

template<typename E>

E CircularQueue<E>::back() const {

if (empty())

error("원형 큐 공백 에러");

else

return queue[\_\_rear];

}

//size: 큐의 크기(항목의 개수)

template<typename E>

int CircularQueue<E>::size() const {

return \_\_count;

}

//empty: 큐의 공백 여부 확인

template<typename E>

bool CircularQueue<E>::empty() const {

return \_\_front == \_\_rear;

}

//full: 큐의 포화 상태 확인

template<typename E>

bool CircularQueue<E>::full() const {

return \_\_count == queueMAXSIZE;

}

//printQueue: 큐의 전체 데이터 항목 출력

template<typename E>

void CircularQueue<E>::printQueue() const {

if (empty())

error("원형 큐 공백 에러");

else {

if (\_\_rear > \_\_front) {

for (int i = \_\_front + 1; i <= \_\_rear; i++) {

cout.width(3);

cout << queue[i];

}

}

else if (\_\_rear < \_\_front) {

for (int i = \_\_front + 1; i < queueMAXSIZE; i++) {

cout.width(3);

cout << queue[i];

}

for (int i = 0; i <= \_\_rear; i++) {

cout.width(3);

cout << queue[i];

}

}

cout << endl;

}

}

inline void error(const char\* message) {

cout << message;

exit(100);

}

int main(void)

{

int num, choice;

CircularQueue<int> CQ= CircularQueue<int>();// 빈 큐 생성

while (true) {

system("cls");

cout << " 20215229 이채훈 " << endl;

cout << "\n ### 큐 구현: 1차원 배열 ### \n" << endl;

cout << "1) 데이터 삽입(enQueue): " << endl;

cout << "2) 데이터 삭제(deQueue): " << endl;

cout << "3) 전체 출력" << endl;

cout << "4) 프로그램 종료 \n" << endl;

cout << "메뉴 선택 : ";

cin >> choice;

switch (choice) {

case 1: cout << "\n삽입 할 데이터 입력: ";

cin >> num;

CQ.push(num);

break;

case 2: cout << "삭제 된 데이터: " << CQ.pop() << endl;

break;

case 3: CQ.printQueue();

cout << "Front: " << CQ.front() << ", Rear: " << CQ.back() << ", Size: "<<CQ.size()<<endl;

break;

case 4: cout << "프로그램 종료..." << endl;

return 0;

default: cout << "잘못 선택 하셨습니다." << endl;

}

system("pause");

}

return 0;

}

**# 연구조사**

**# Circular Queue**

생성자&소멸자

큐함수- push(enqueue), pop(dequeue)

탐색-empty(), full()

출력-printStack(), front(), back(), size()

**1. empty**

큐가 비어있는지 확인한다. front와 rear가 같으면 큐가 비었다고 판단한다.

**2. full**

큐가 꽉 찼는지 확인한다. 선형 큐의 경우, rear+1의 값이 queueMaXSIZE와 같은지 확인하면 되지만,

원형 큐는 같은 방법으로 할 경우, front = rear인 경우가 생겨, empty와 구분할 수 없다.

따라서, 별도의 count 변수를 유지하여 데이터의 갯수를 세고, count == queueMaXSIZE와 같은지 확인해야한다.

**3. printQueue**

큐의 모든 데이터를 확인한다.

rear > front인 경우는 선형 큐와 같은 상황이므로 front+1부터 rear까지 출력하면 되지만,

rear < front인 경우는 queue의 마지막 칸까지 채워진 상태에서, 비어있는 앞칸에 데이터를 삽입하는 경우이므로 두 부분을 나누어 출력해준다.

**4. push(enQueue)**

rear에 data를 입력한다.

(rear+1)%queueMAXSIZE의 의미 : rear값을 반드시 0~9 사이값으로 만든다.

이는 rear가 queueMAXSIZE-1 의 인덱스에 도달할 경우, rear를 다시 0부터 시작하도록 해주며, 앞칸부터 다시 삽입할 수 있도록 해준다

-> 꽉 차있는지 확인

-> 꽉 차지 않은 경우 rear = (rear+1)%queueMAXSIZE로 삽입할 자리 선정

-> 이 후 삽입 실행

-> count 갯수 증가

**5. pop(deQueue)**

front에서 data를 삭제한다.

-> 비어있는지 확인

-> 비어있지 않은 경우, count 갯수 감소 후 , 현재 front의 다음칸을 반환

**06-7. 3차 실습보고서\_연습문제#03\_원형 큐 구현(순차 자료구조)ver.서**

**# 코드**

#include <iostream>

using namespace std;

#define queueMAXSIZE 7

template <typename E>

class CircularQueue {

private:

E queue[queueMAXSIZE] = { 0 };

int frontIndex, rearIndex, count;

public:

//생성자 & 소멸자

CircularQueue();

~CircularQueue();

//push, pop,front, back

void push(const E& e);

void pop();

E front() const;

E back() const;

//size, empty, full

int size() const;

bool empty() const;

bool full() const;

//printQueue

void printQueue() const;

};

template<typename E>

CircularQueue<E>::CircularQueue()

: frontIndex(-1), rearIndex(-1), count(0) {}

template<typename E>

CircularQueue<E>::~CircularQueue() {}

//push(enQueue): 큐에 데이터 항목 삽입(맨 마지막 원소)

template<typename E>

void CircularQueue<E>::push(const E& e) {

if (full()) {

cout << "큐가 포화 상태입니다." << endl;

//return;

}

else{

rearIndex = (rearIndex + 1) % queueMAXSIZE;

queue[rearIndex] = e;

count++;

}

}

//pop(deQueue): 큐에서 맨 첫 번째로 삽입된 데이터 항목 삭제

template<typename E>

void CircularQueue<E>::pop() {

if (empty()) {

cout << "큐가 공백 상태입니다." << endl;

//return;

}

else {

frontIndex = (frontIndex + 1) % queueMAXSIZE;

queue[frontIndex] = 0;

count--;

}

}

//front(peek): 큐에서 맨 첫 번째로 삽입된 데이터 항목 확인

template<typename E>

E CircularQueue<E>::front() const {

return queue[(frontIndex+1)%queueMAXSIZE];

}

//back(peek): 큐에서 맨 마지막에 삽입된 데이터 항목 확인

template<typename E>

E CircularQueue<E>::back() const {

return queue[rearIndex];

}

//size: 큐의 크기(항목의 개수)

template<typename E>

int CircularQueue<E>::size() const {

return count;

}

//empty: 큐의 공백 여부 확인

template<typename E>

bool CircularQueue<E>::empty() const {

return count == 0;

}

//full: 큐의 포화 상태 확인

template<typename E>

bool CircularQueue<E>::full() const {

return (rearIndex + 1) % queueMAXSIZE == frontIndex;

}

//printQueue: 큐의 전체 데이터 항목 출력

template<typename E>

void CircularQueue<E>::printQueue() const {

cout << "\n QUEUE [";

for (int i = 0; i < queueMAXSIZE; i++) {

cout.width(3);

cout << queue[i];

}

cout << "]" << endl;

}

int main(void)

{

int num, choice;

CircularQueue<int> CQ;

while (true) {

system("cls");

cout << " 20215229 이채훈 " << endl;

cout << "\n ### 큐 구현: 1차원 배열 ### \n" << endl;

cout << "1) 데이터 삽입(enQueue): " << endl;

cout << "2) 데이터 삭제(deQueue): " << endl;

cout << "3) 전체 출력" << endl;

cout << "4) 프로그램 종료 \n" << endl;

cout << "메뉴 선택 : ";

cin >> choice;

switch (choice) {

case 1:

cout << "\n삽입 할 데이터 입력: ";

cin >> num;

CQ.push(num);

break;

case 2:

cout << "삭제 된 데이터: " << CQ.front() << endl;

CQ.pop();

break;

case 3:

cout << "원형 큐 전체 원소 출력: "; CQ.printQueue();

cout << "\nFront: " << CQ.front() << ", Rear: " << CQ.back() << ", Size: " << CQ.size() << endl;

break;

case 4:

cout << "프로그램 종료..." << endl;

return 0;

default:

cout << "잘못 선택 하셨습니다." << endl;

}

system("pause");

}

return 0;

}

**06-8. 3차 실습보고서\_연습문제#03\_원형 큐 구현(순차 자료구조)ver.옹**

**# 코드**

#include <iostream>

#define queueMAXSIZE 7

using namespace std;

template <typename E>

class CircularQueue

{

E queue[queueMAXSIZE];

int front, rear, count;

public:

CircularQueue();//생성자

~CircularQueue();//소멸자

void push(const E& e);//삽입

E pop();//삭제

E printfront() const;//프론트 원소 출력

E printback() const; //큐 제일 뒤에 있는 값 반환

int size() const;//큐의 크기 출력

bool empty() const; //큐 빈 여부 확인

bool full() const; //큐 포화 여부 확인

void printQueue() const; //큐 배열 원소 출력

};

template <typename E>

CircularQueue <E> ::CircularQueue() : front(0), rear(0), count(0) {}

template <typename E>

CircularQueue<E>::~CircularQueue() {} // 소멸자

template <typename E>

void CircularQueue<E>::push(const E& e)

{

if (full())

{

cout << "큐 포화 상태로 푸쉬 불가" << endl;

return;

}

queue[rear] = e;

rear = (rear + 1) % queueMAXSIZE;

++count;

}

template<typename E>

E CircularQueue<E>::pop()

{

if (empty())

{

cout << "큐가 비어 있음 " << endl;

return 0;

}

E pop\_data = queue[front];

front = (front + 1) % queueMAXSIZE;

--count;

return pop\_data;

}

template<typename E>

E CircularQueue<E>::printfront() const

{

if (empty())

{

cout << "큐가 비어있습니다." << endl;

return 0;

}

return queue[front];

}

template<typename E>

E CircularQueue<E>::printback() const

{

if (empty())

{

cout << "큐가 비어있습니다." << endl;

return 0;

}

return queue[(rear - 1 + queueMAXSIZE) % queueMAXSIZE];

}

template<typename E>

int CircularQueue<E>::size() const

{

return count;

}

template<typename E>

bool CircularQueue<E>::empty() const

{

return (count == 0);

}

template<typename E>

bool CircularQueue<E>::full() const

{

return (count == queueMAXSIZE);

}

template<typename E>

void CircularQueue<E>::printQueue() const

{

if (empty())

{

cout << "큐가 비어 있습니다" << endl;

return;

}

if (full())

{

cout << "큐가 꽉 찼습니다" << endl;

return;

}

int i = front;

for (int j = 0; j < count; j++)

{

cout << queue[i] << " ";

i = (i + 1) % queueMAXSIZE;

}

cout << endl;

}

int main()

{

CircularQueue<int> q; //큐 클래스 선언

int menu;

cout << endl << "### 큐 구현: 1차원 배열 ###" << endl << endl;

cout << "1) 데이터 삽입(enQueue):" << endl;

cout << "2) 데이터 삭제(deQueue):" << endl;

cout << "3) 전체 출력" << endl;

cout << "4) 프로그램 종료" << endl << endl;

while (true)

{

cout << "메뉴 선택: "; cin >> menu;

switch (menu)

{

case 1: //1) 데이터 삽입

int input;

cout << "삽입할 데이터 입력: "; cin >> input;

q.push(input);

break;

case 2: //2) 데이터 삭제

q.pop();

break;

case 3: //3) 데이터 출력

q.printQueue();

break;

case 4: //4) 프로그램 종료

return 0;

}

cout << endl;

}

return 0;

}

**# 연구조사**

**\* count 변수를 사용하는 이유**

원형 큐에 empty한 큐와 full한 큐를 판정할 때 front=rear+1이라는 조건문을 사용하여 판별하는데, 이러면 아래의 그림과 같이 empty와 full 두 상태의 구분이 어려우므로 count 함수를 이용해 empty와 full한 큐를 분별해야 한다.

도표, 원, 스크린샷, 라인이(가) 표시된 사진

자동 생성된 설명

위의 코드의 CircularQueue 클래스는 1차원 배열을 사용하여 원형 큐를 구현하고, 큐의 기본 동작을 수행하는 멤버 함수이다.

front, rear, count 멤버 변수를 이용하여 큐의 상태를 관리하며, 이를 이용하여 삽입, 삭제, 출력 등의 연산을 한다.

또한 printQueue 함수는 큐의 현재 상태를 출력하며, size, empty, full 함수들은 각각 큐의 크기, 포화 및 공백 여부를 반환한다.

스택을 배열로 구현할 때 가장 중요한 것은 변수 top이다.

top이라는 변수는 스택 배열의 가장 큰 숫자를 가진 인덱스를 의미하는 변수로, 스택으로 치면 가장 위쪽의 원소를 가리키는 인덱스 변수라고 볼 수 있다.

스택의 삽입 시 ‘st[++top]=e’과 같이 전위 증가 연산자를 사용하여 top 변수를 증가시킨 이유는, 언제나 가장 나중에 추가된 원소를 가리키는 인덱스인 top 변수를 이용해 추가로 원소를 삽입할 때, top의 위치도 같이 바꾼 상태에서, 업데이트 된 top의 위치에 해당하는 자리에 새 원소를 추가해야 하기 때문이다.