Data Structure

실습 3

0. 이번 주 실습 내용

- 과제2 기간 연장 (4월 9일 11시 59분까지)
- Stack
  - Stack 개념 및 활용 예
  - Stack 구조 (Array List / Linked List)
  - Stack 실습 (Linked List)
- Queue
  - Queue 개념 및 활용 예
  - Queue 구조 (Array List / Linked List)
  - Circular Queue 실습 (Array List)

1. Stack

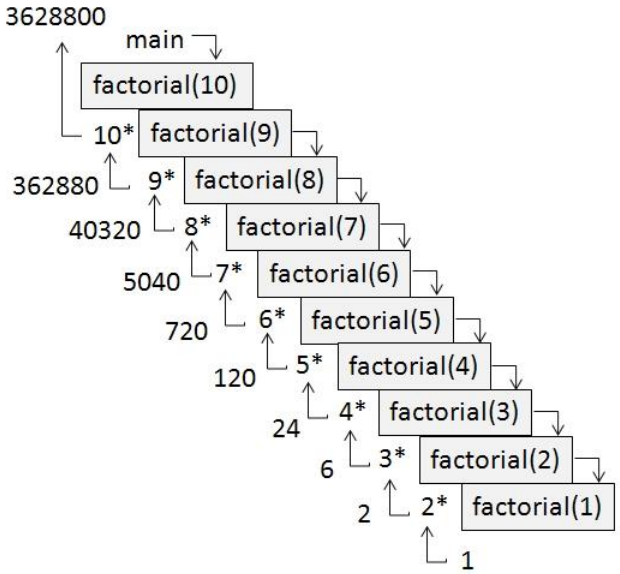
- **정의: 한 쪽 끝에서만 데이터를 입/출력 할 수 있는 자료구조**
  - 가장 마지막에 넣은 것이 먼저 나오는 형태: **LIFO** (Last In First Out)
  - 가장 마지막 부분(혹은 가장 윗부분)을 가리키는 포인터형 변수(top 또는 head)가 있는 것이 특징

- 기본 연산 단위
  - push : 데이터를 넣는 연산
  - pop : 데이터를 꺼내는 연산
  - top : 현재 스택에 맨 위에 있는 데이터에 접근하는 연산

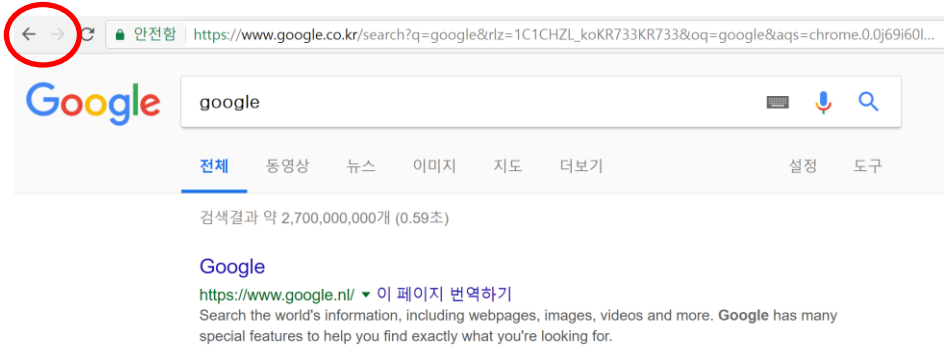
- 구현 형태: Array List 또는 Linked List로 구현
  - 본 실습에서는 Linked List 방식을 이용

1. Stack

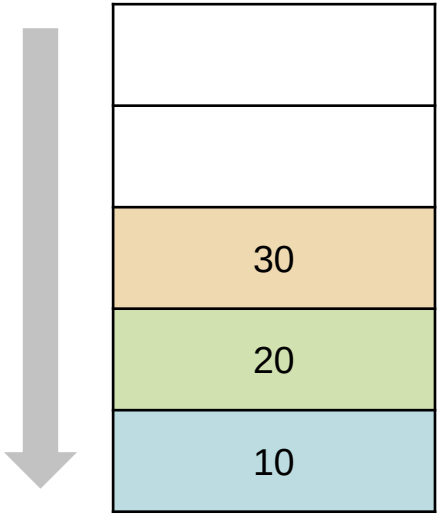
- Stack의 활용
  - Function call stack



- Stack의 활용
  - 뒤로 가기 / 앞으로 가기



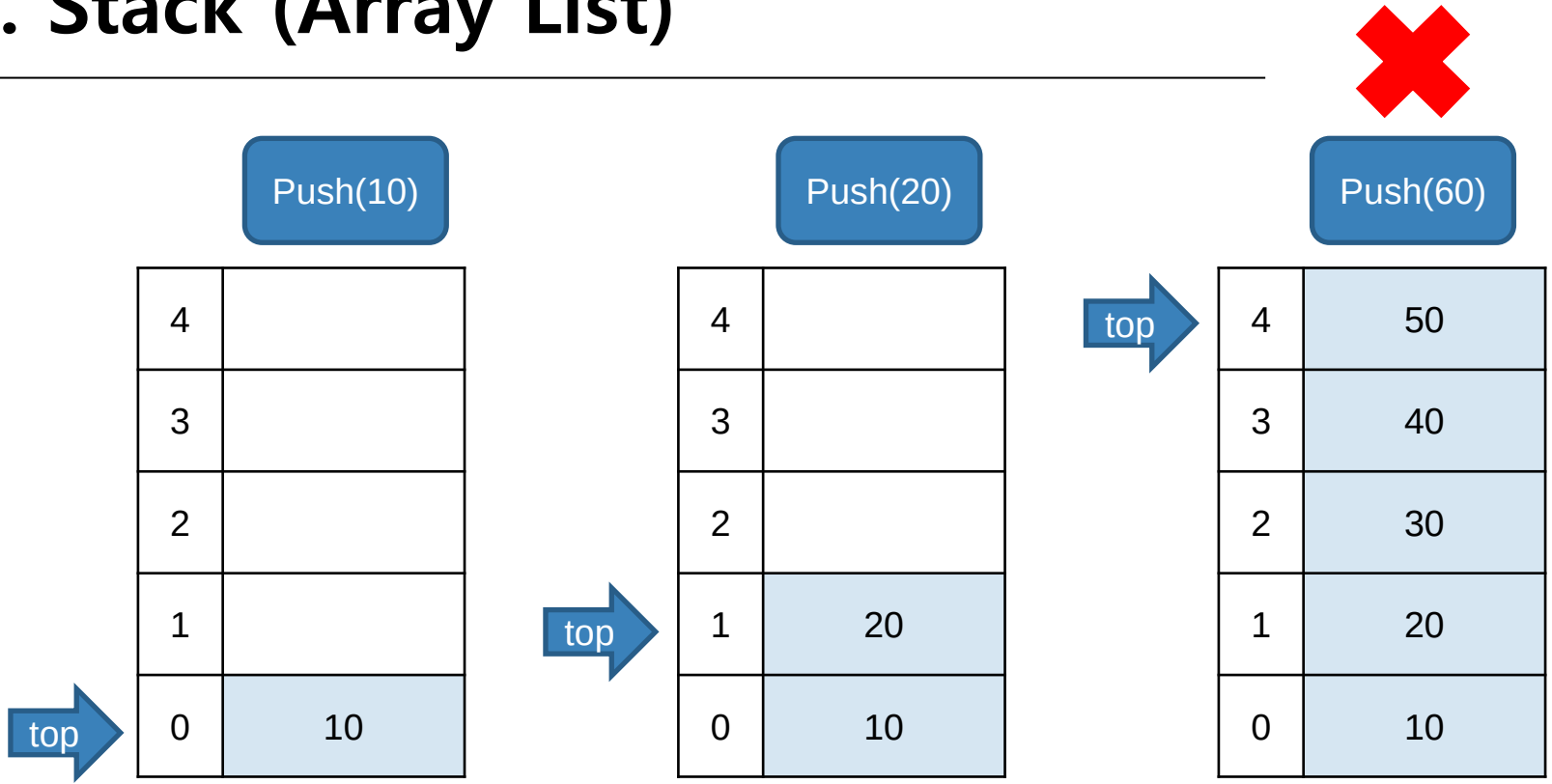
1. Stack



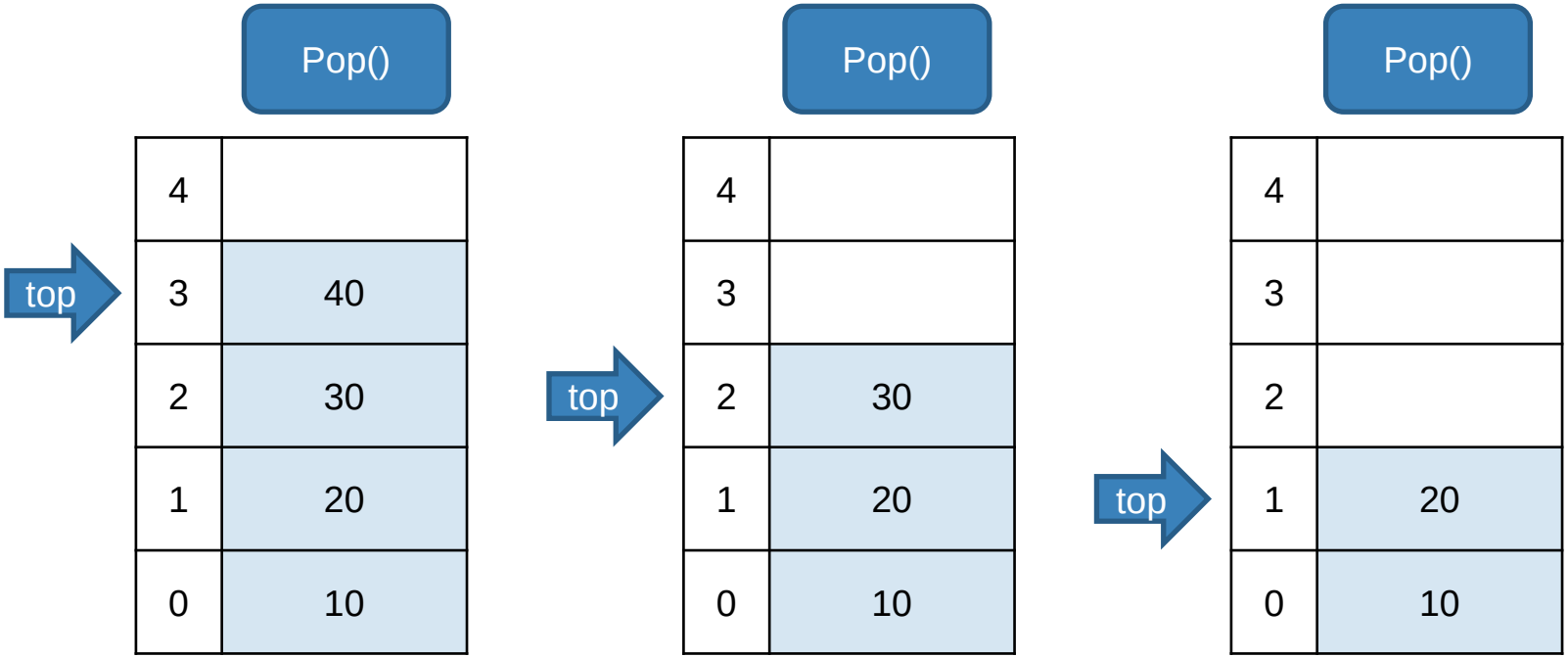
자료를 한 방향으로만 쌓는 자료구조(Stack)의
기본 연산 및 구조

Push() : 자료를 스택에 추가
Pop() : 스택에서 자료를 꺼냄
Top() : 스택의 현재 위치 자료에 접근

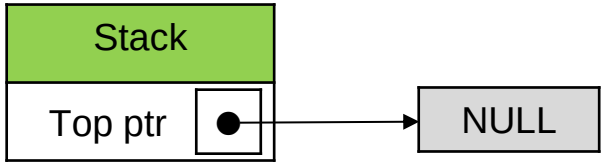
1. Stack (Array List)

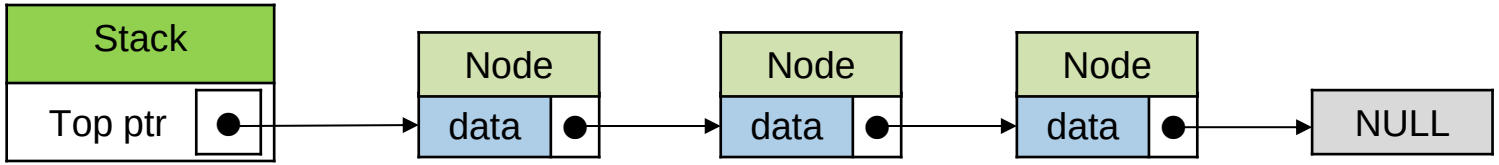


1. Stack (Array List)

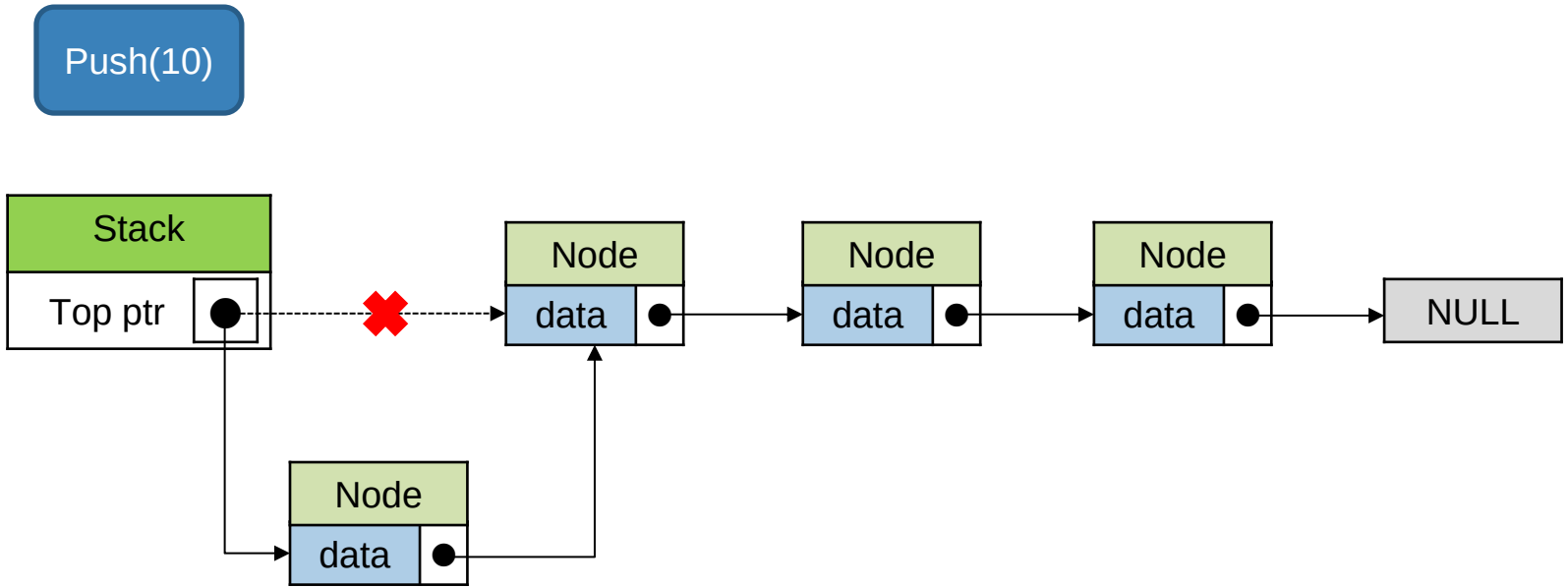


1. Stack (Linked List)

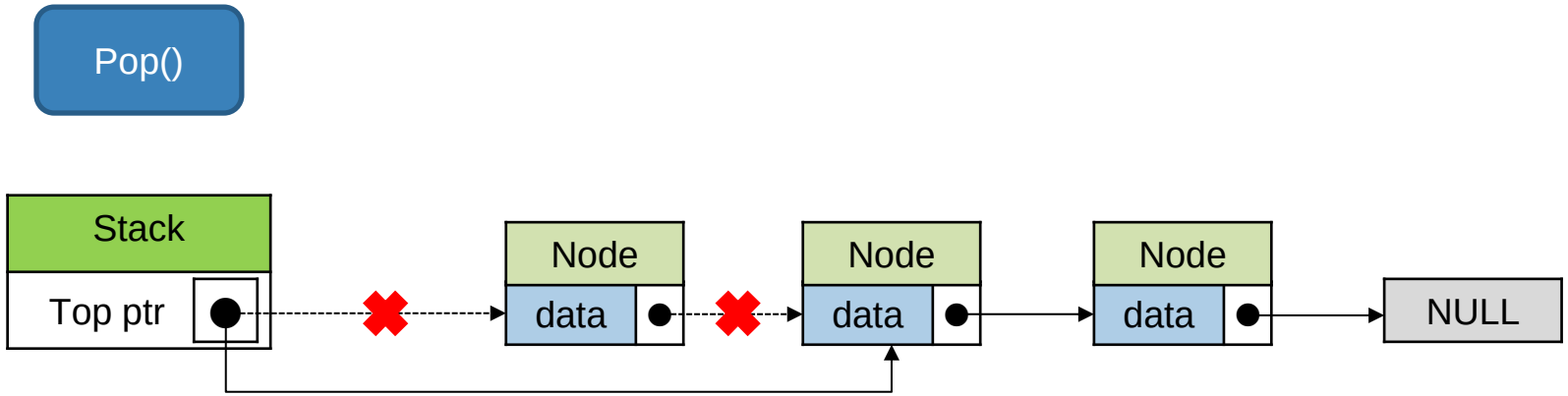




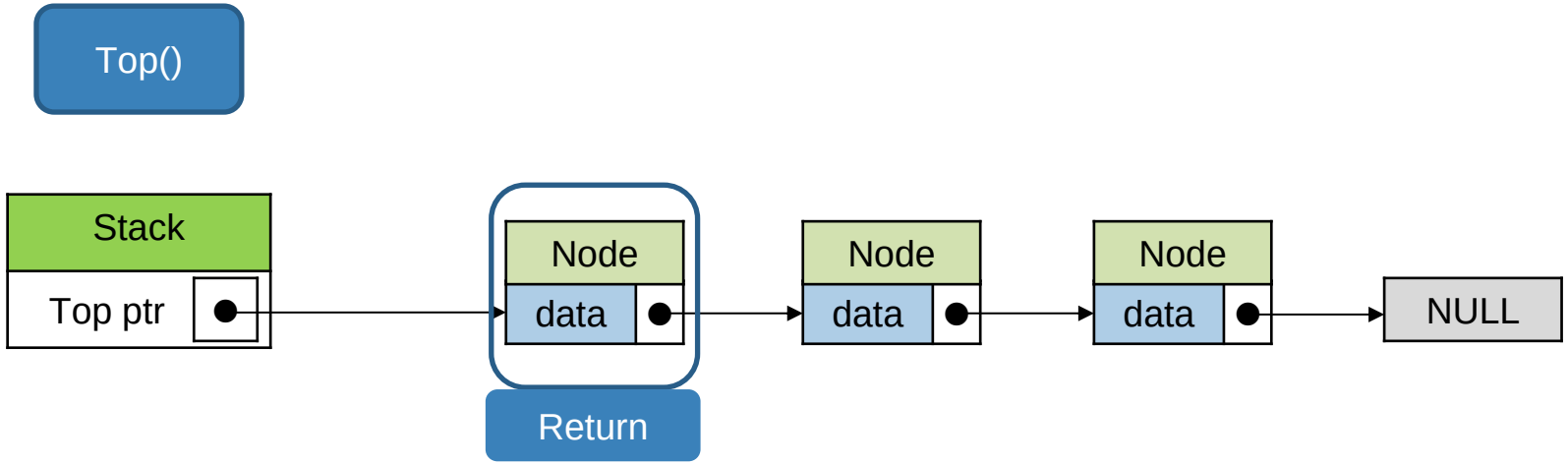
1. Stack (Linked List)



1. Stack (Linked List)

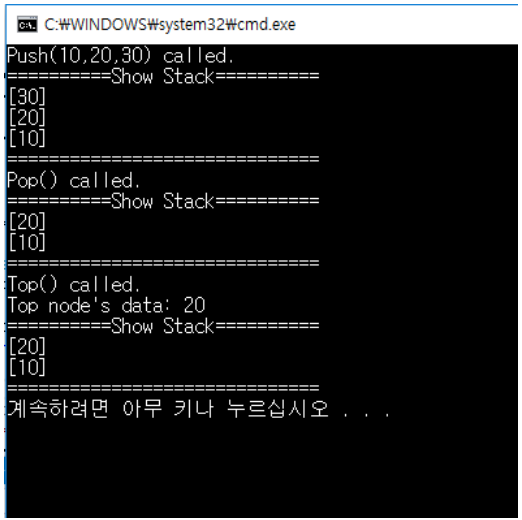


1. Stack (Linked List)



1. Stack (Linked List) - 실습

- Stack 구조체 선언
- main 함수 정의




```c
#include <stdio.h>
#include <stdlib.h>
#define TRUE 1
#define FALSE 0

//Stack 구조체 선언 (Linked List)
typedef struct StackNode {
    int data;
    struct StackNode* next;
}StackNode;

//Stack 관련 함수
void pushLinkedStack(StackNode** top, int data);
StackNode* popLinkedStack(StackNode** top);
void showLinkedStack(StackNode* top);
StackNode* topLinkedStack(StackNode* top);
void deleteLinkedStack(StackNode** top);
int isEmpty(StackNode* top);

int main()
{
    //가장 윗 부분을 가리키는 top 포인터 선언
    StackNode* top = NULL;
    StackNode* pNode;

    printf("Push(10,20,30) called.₩n");
    pushLinkedStack(&top, 10);
    pushLinkedStack(&top, 20);
    pushLinkedStack(&top, 30);
    showLinkedStack(top);

    printf("Pop() called.₩n");
    pNode = popLinkedStack(&top);
    if (pNode)
    {
        free(pNode);
        showLinkedStack(top);
    }

    printf("Top() called. ₩n");
    pNode = topLinkedStack(top);
    if (pNode)
        printf("Top node's data: %d₩n", pNode->data);
    else
        printf("The Stack is empty₩n");
    showLinkedStack(top);

    deleteLinkedStack(&top);
    return 0;
}
```

1. Stack (Linked List) - 실습

- isEmpty 함수 정의
  :Stack이 비어 있는지 검사

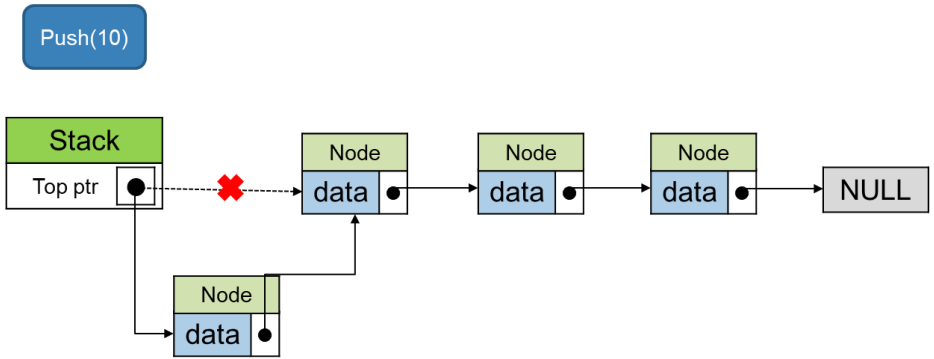
```
52  int isEmpty(StackNode* top)
53  {
54      if (top == NULL)
55          return TRUE;
56      else
57          return FALSE;
58  }
```

- Stack show 함수 정의
  :top 부터 stack의 끝 Node까지 data를 출력

```
100 void showLinkedStack(StackNode* top)
101 {
102     StackNode *pNode = NULL;
103     if (isEmpty(top))
104     {
105         printf("the Stack is empty₩n");
106         return;
107     }
108
109     pNode = top;
110     printf("==========Show Stack==========₩n");
111     while (pNode != NULL)
112     {
113         printf("[%d]₩n", pNode->data);
114         pNode = pNode->next;
115     }
116     printf("==============================₩n");
117 }
118
```

1. Stack (Linked List) - 실습

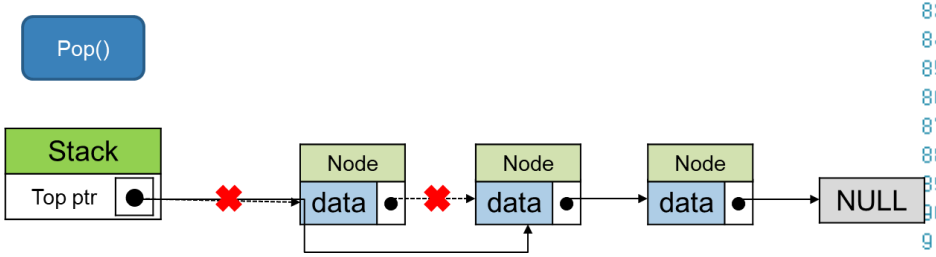
- Stack push 함수 정의
  :데이터를 추가하여 top에 위치시킴




```
52  void pushLinkedStack(StackNode** top, int data)
53  {
54      StackNode *pNode = NULL;
55
56      //넣을 Stack Node를 할당하고 data값을 저장
57
58
59
60
61      //만일 Stack이 empty일 경우 바로 top으로 지정
62
63
64      //Stack에 node가 하나라도 있는 경우 이어주기
65      else
66      {
67          //넣을 Stack Node의 다음 노드가 현재의 top Node를 가리키고
68          //top Node를 넣을 Stack Node로 지정
69
70
71      }
72  }
```

1. Stack (Linked List) - 실습

- Stack pop 함수 정의
  :top이 가리키고 있는 데이터를 삭제




```
82  StackNode* popLinkedStack(StackNode** top)
83  {
84      StackNode *pNode = NULL;
85
86      //Stack이 비어있는지 검사
87
88
89
90
91
92      //pNode에 top이 가리키는 Node를 지정하고 top은 그 다음 Node로 지정
93
94
95
96      return pNode;
97
98  }
```

1. Stack (Linked List) - 실습

- Stack top 함수 정의
  :top이 가리키고 있는 노드를 반환

```
122  StackNode* topLinkedStack(StackNode* top)
123  {
124      StackNode *pNode = NULL;
125
126      if (!isEmpty(top))
127          pNode = top;
128      return pNode;
129  }
```

- Stack delete 함수 정의
  :top 부터 stack의 끝 Node까지 메모리 해제

```
131  void deleteLinkedStack(StackNode** top)
132  {
133      StackNode *pNode = NULL, *pDelNode = NULL;
134      pNode = *top;
135
136      //pNode를 한 칸씩 이동하면서 메모리 해제
137      while (pNode != NULL)
138      {
139          pDelNode = pNode;
140          pNode = pNode->next;
141          free(pDelNode);
142      }
143      *top = NULL;
144  }
```

2. Queue

- **정의: 양 쪽 끝에서 데이터를 입/출력 할 수 있는 자료구조**
  - 가장 먼저 넣은 것이 먼저 나오는 형태: **FIFO** (First In First Out)
  - 처음 부분과 마지막 부분을 가리키는 포인터형 변수(front, rear)가 있는 것이 특징

- 기본 연산 단위
  - enqueue (add queue) : 데이터를 넣는 연산
  - dequeue (delete queue) : 데이터를 꺼내는 연산

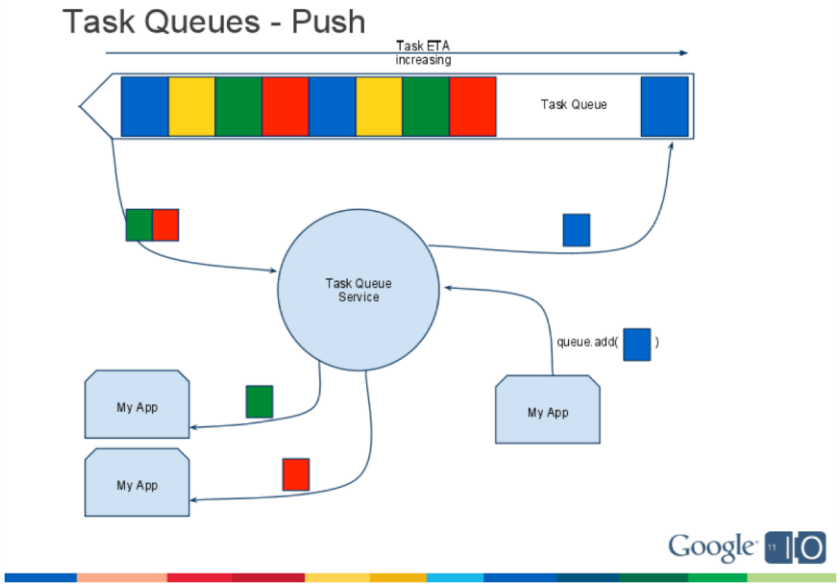
- 구현 형태: Array List 또는 Linked List로 구현
  - 본 실습에서는 Array List 방식을 이용하여 Circular queue를 구현

2. Queue

- Queue의 활용
  - Video Streaming

Source: [Gissler et al. 2020]

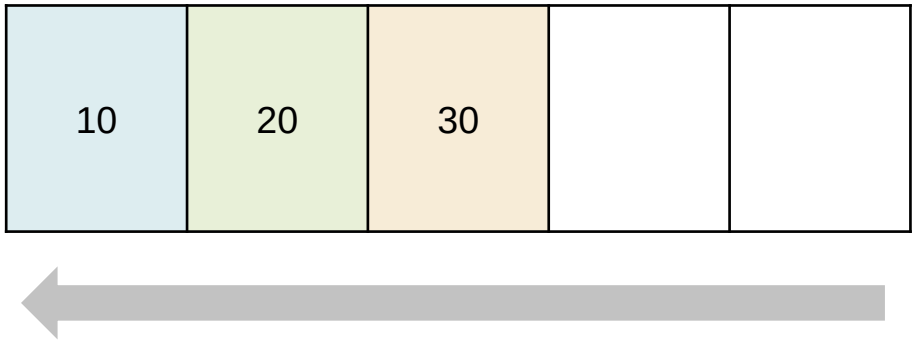
- Queue의 활용
  - CPU Scheduling



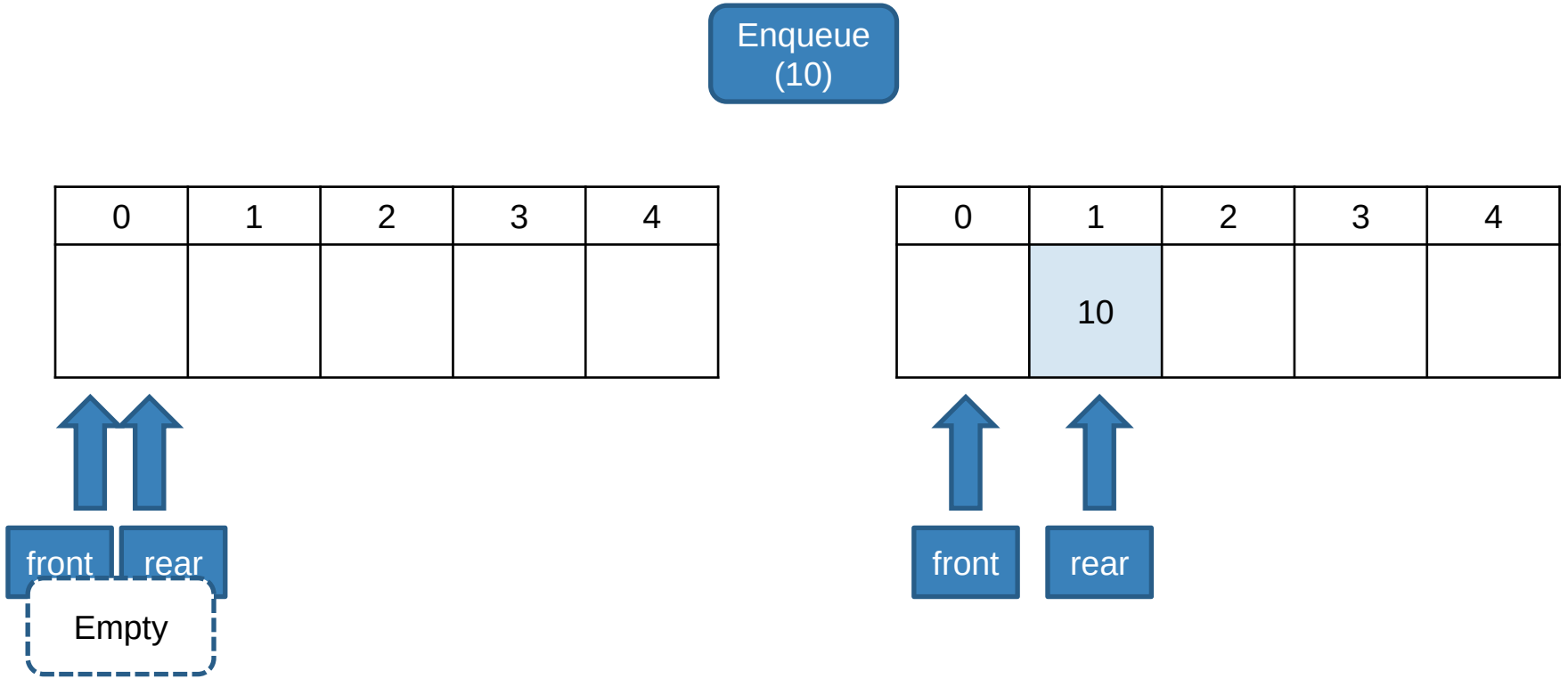
2. Queue

자료를 한 쪽으로 넣고 반대편으로 빼는
자료구조(Queue)의 기본 연산 및 구조

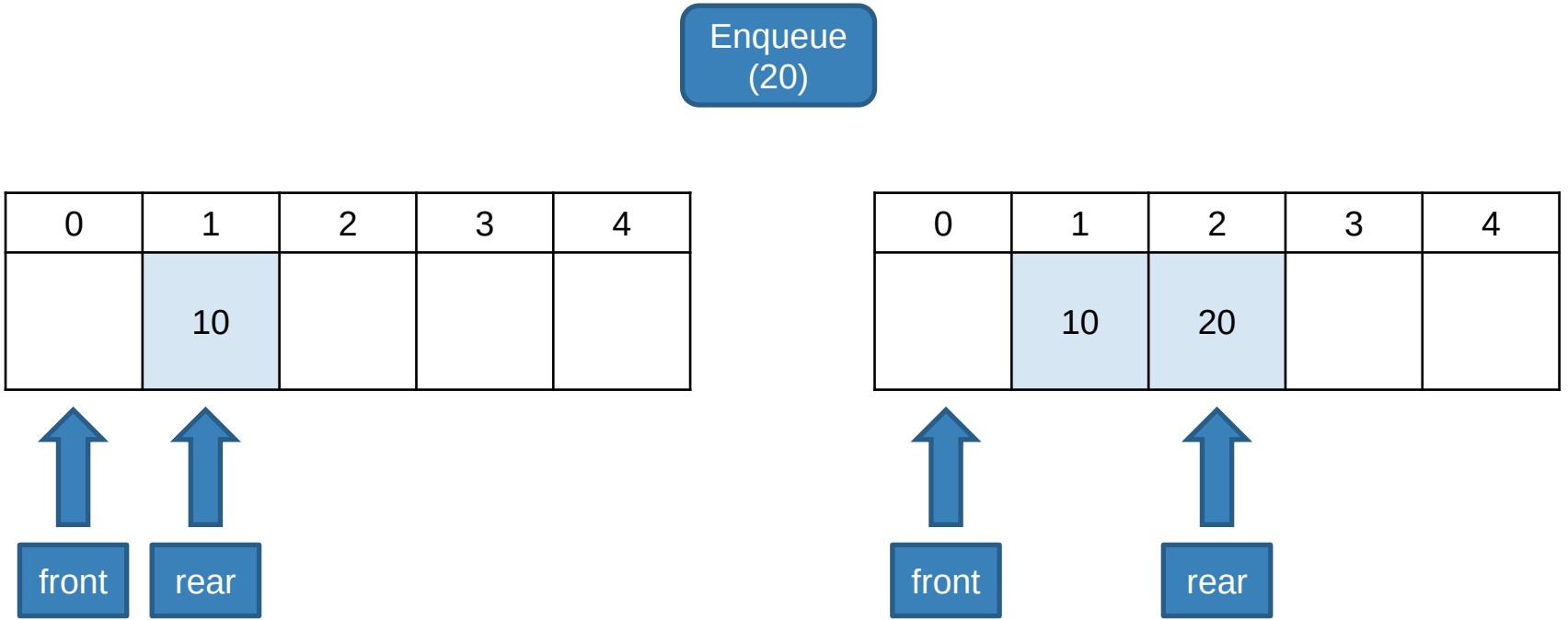
Enqueue() : 자료를 큐에 추가
Dequeue() : 큐에서 자료를 꺼냄



2. Queue (Array List)

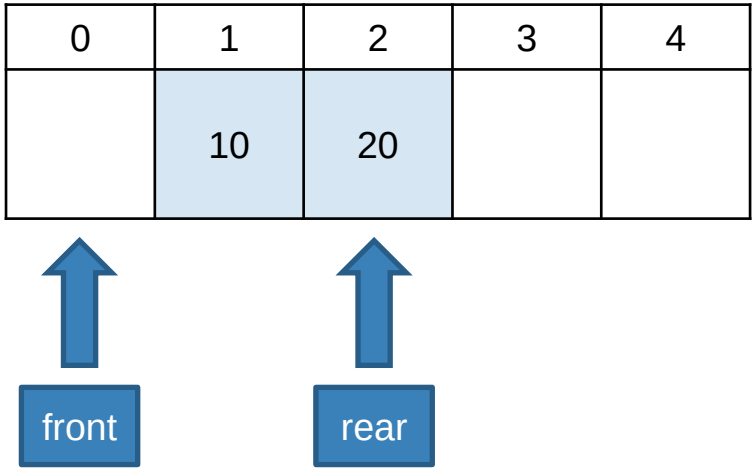


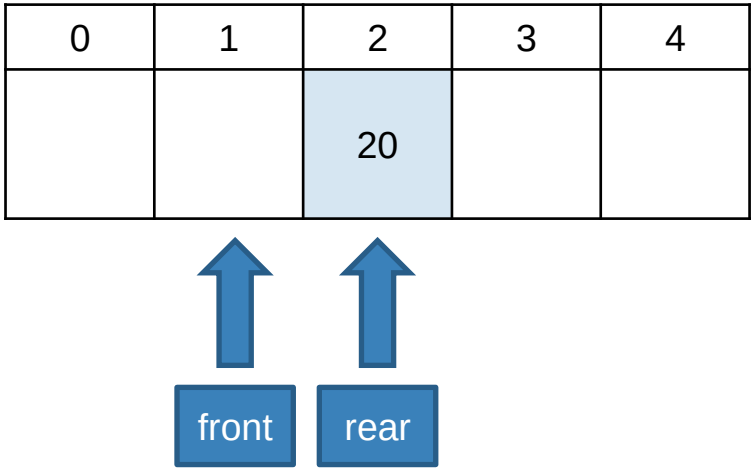
2. Queue (Array List)



2. Queue (Array List)

Dequeue
()

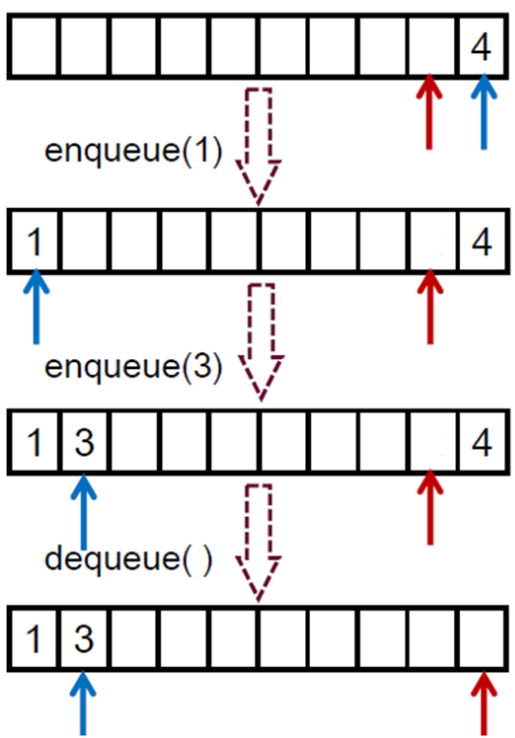


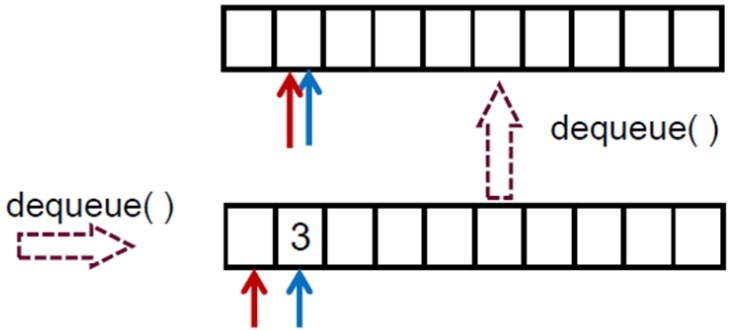


2. Queue (Array List)

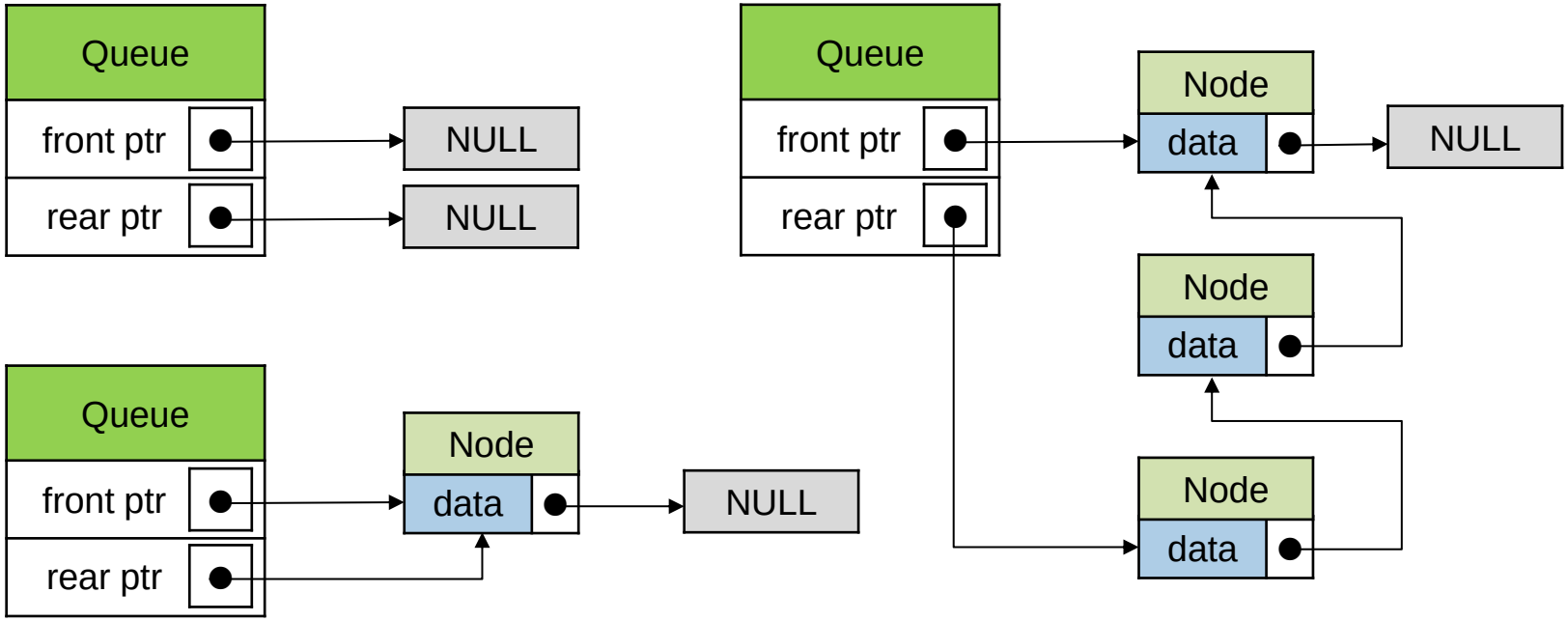
- **Circular Queue**

  - Array List로 구현된 Queue의 한계점을 극복하고자 만들어짐

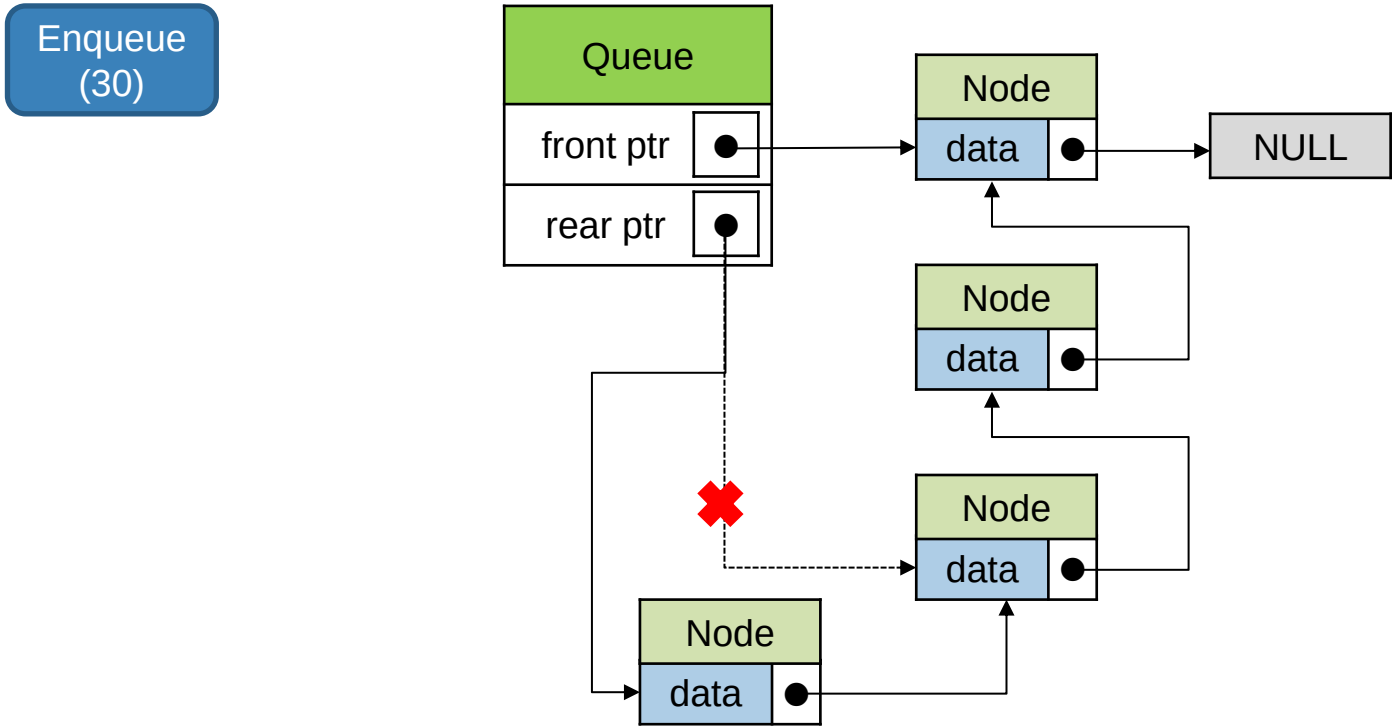




2. Queue (Linked List)

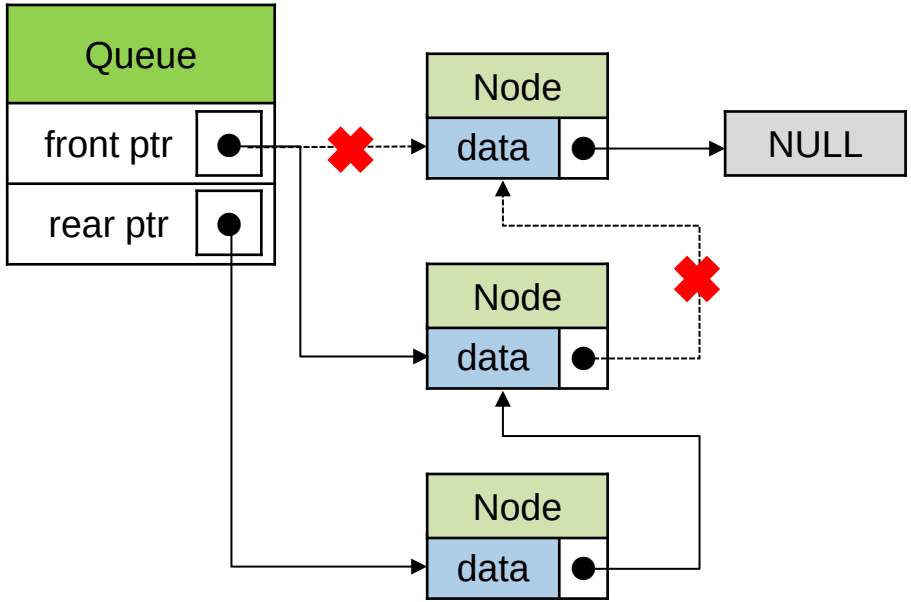


2. Queue (Linked List)



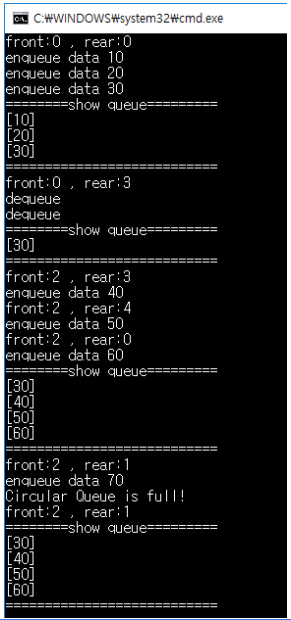
2. Queue (Linked List)

Dequeue
()



2. Circular Queue (Array List) - 실습

- Queue 구조체 선언
- main 함수 정의




```c
#include <stdio.h>
#include <stdlib.h>

#define Capacity 5
#define TRUE 1
#define FALSE 0

//Circular Queue 구조체 선언
typedef struct circularQueue {
    int data[Capacity]; //Array List data 선언
    int front;          //Circular Queue의 앞부분
    int rear;           //Circular Queue의 마지막 부분
}cQueue;

//Circular Queue 관련 함수
cQueue* createCircularQueue();
void enqueue(cQueue* cQueue, int data);
int isFull(cQueue* cQueue);
void showQueue(cQueue* cQueue);
int isEmpty(cQueue* cQueue);
void dequeue(cQueue* cQueue);

int main()
{
    cQueue* cQueue;
    //Circular Queue 초기화
    cQueue = createCircularQueue();
    printf("front:%d , rear:%d₩n", cQueue->front, cQueue->rear);

    printf("enqueue data 10₩n");
    printf("enqueue data 20₩n");
    printf("enqueue data 30₩n");
    enqueue(cQueue, 10);
    enqueue(cQueue, 20);
    enqueue(cQueue, 30);
    showQueue(cQueue);
    printf("front:%d , rear:%d₩n", cQueue->front, cQueue->rear);

    printf("dequeue₩n");
    printf("dequeue₩n");
    dequeue(cQueue);
    dequeue(cQueue);
    showQueue(cQueue);
    printf("front:%d , rear:%d₩n", cQueue->front, cQueue->rear);

    printf("enqueue data 40₩n");
    enqueue(cQueue, 40);
    printf("front:%d , rear:%d₩n", cQueue->front, cQueue->rear);
    printf("enqueue data 50₩n");
    enqueue(cQueue, 50);
    printf("front:%d , rear:%d₩n", cQueue->front, cQueue->rear);
    printf("enqueue data 60₩n");
    enqueue(cQueue, 60);
    showQueue(cQueue);
    printf("front:%d , rear:%d₩n", cQueue->front, cQueue->rear);
    printf("enqueue data 70₩n");
    enqueue(cQueue, 70);
    printf("front:%d , rear:%d₩n", cQueue->front, cQueue->rear);
    showQueue(cQueue);
    return 0;
}
```

2. Circular Queue (Array List) - 실습

- Queue 초기화 함수 정의
  :비어 있는 Queue을 생성

```
59  cQueue* createCircularQueue( )
60  {
61      cQueue* pCQueue = NULL;
62      int i;
63      //메모리상에 Queue를 할당시키고 이를 반환
64      pCQueue = (cQueue *)malloc(sizeof(cQueue));
65      pCQueue->front = 0;
66      pCQueue->rear = 0;
67
68      return pCQueue;
69  }
```

- Queue show 함수 정의
  :데이터를 Queue의 front에서 rear까지 출력

```
99  void showQueue(cQueue* cQueue)
100 {
101     int i;
102     if (isEmpty(cQueue) == TRUE)
103     {
104         printf("Circular Queue is Empty!₩n");
105         return;
106     }
107     printf("========show queue=========₩n");
108     for (i = cQueue->front+1; i != cQueue->rear; i=(i+1)%Capacity)
109         printf("[%d]₩n", cQueue->data[i]);
110     printf("[%d]₩n", cQueue->data[i]);
111     printf("============================₩n");
112 }
```

2. Circular Queue (Array List) - 실습

- Queue enqueue 함수 정의
  :데이터를 Queue의 rear 부분에 추가
  Circular Queue처럼 작동하도록 %연산을 활용

```
76  void enqueue(cQueue* cQueue, int data)
77  {
78      if (
79      {
80          printf("Circular Queue is full!₩n");
81          return;
82      }
83      //Circular Queue의 뒷 부분에 data를 추가
84
85
86  }
```

- Queue dequeue 함수 정의
  :Queue의 front 부분에 있는 데이터를 제거
  Circular Queue처럼 작동하도록 %연산을 활용

```
88  void dequeue(cQueue* cQueue)
89  {
90      if (
91      {
92          printf("Circular Queue is Empty!₩n");
93          return;
94      }
95      //Circular Queue의 앞 부분을 다음 칸으로 이동
96
97  }
98
```

2. Circular Queue (Array List) - 실습

- Queue isFull 함수 정의
  :Queue의 데이터가 가득 차 있는지 확인

```
107  int isFull(cQueue* cQueue)
108  {
109      if ((cQueue->rear + 1) % Capacity == cQueue->front)
110          return TRUE;
111      else
112          return FALSE;
113  }
```

- Queue isEmpty 함수 정의
  :Queue의 데이터가 비어 있는지 확인

```
115  int isEmpty(cQueue* cQueue)
116  {
117      if (cQueue->front == cQueue->rear)
118          return TRUE;
119      else
120          return FALSE;
121  }
```