컴퓨터공학 All in One

C/C++ 문법, 자료구조 및 심화 프로젝트 (나동빈)
제 24강 - 큐

Fast campus

학습 목표

큐

1) 큐 자료구조의 개념과 활용 방법에 대해서 이해합니다.
2) C언어를 이용해 큐 자료구조를 직접 구현할 수 있습니다.

큐

큐

1) 큐(Queue)는 뒤쪽으로 들어가서 앞쪽으로 나오는 자료 구조(Data Structure)입니다.
2) 이러한 특성 때문에 스케줄링, 탐색 알고리즘 등에서 다방면으로 활용됩니다.

- PUSH: 큐에 데이터를 넣습니다.
- POP: 큐에서 데이터를 빼냅니다.

큐

큐

1) 큐(Queue)는 뒤쪽으로 들어가서 앞쪽으로 나오는 자료 구조(Data Structure)입니다.

PUSH(7) - PUSH(5) - PUSH(4) - POP() - PUSH(6) - POP()

큐

큐

1) 큐(Queue)는 뒤쪽으로 들어가서 앞쪽으로 나오는 자료 구조(Data Structure)입니다.

PUSH(7) - PUSH(5) - PUSH(4) - POP() - PUSH(6) - POP()

큐

큐

1) 큐(Queue)는 뒤쪽으로 들어가서 앞쪽으로 나오는 자료 구조(Data Structure)입니다.

PUSH(7) - **PUSH(5)** - PUSH(4) - POP() - PUSH(6) - POP()

큐

큐

1) 큐(Queue)는 뒤쪽으로 들어가서 앞쪽으로 나오는 자료 구조(Data Structure)입니다.

PUSH(7) - **PUSH(5)** - **PUSH(4)** - POP() - PUSH(6) - POP()

큐

큐

1) 큐(Queue)는 뒤쪽으로 들어가서 앞쪽으로 나오는 자료 구조(Data Structure)입니다.

PUSH(7) - **PUSH(5)** - **PUSH(4)** - **POP()** - PUSH(6) - POP()

큐

큐

1) 큐(Queue)는 뒤쪽으로 들어가서 앞쪽으로 나오는 자료 구조(Data Structure)입니다.

PUSH(7) - **PUSH(5)** - **PUSH(4)** - **POP()** - **PUSH(6)** - POP()

큐

큐

1) 큐(Queue)는 뒤쪽으로 들어가서 앞쪽으로 나오는 자료 구조(Data Structure)입니다.

PUSH(7) - PUSH(5) - PUSH(4) - POP() - PUSH(6) - POP()

큐

큐의 구현

1) 큐(Queue)는 배열을 이용한 구현 방법과 연결 리스트를 이용한 구현 방법으로 나누어 집니다.
2) 가장 기본적인 형태의 자료구조로 구현 난이도는 낮은 편입니다.
3) 먼저 배열을 이용한 구현 방법에 대해서 알아보겠습니다.

큐

배열을 이용한 큐 구현

- 큐의 선언

```c
#include <stdio.h>
#define SIZE 10000
#define INF 99999999

int queue[SIZE];
int front = 0;
int rear = 0;
```

큐

배열을 이용한 큐 구현

- 큐 삽입 함수

```
void push(int data) {
  if (rear >= SIZE) {
    printf("큐 오버플로우가 발생했습니다.\n");
    return;
  }
  queue[rear++] = data;
}
```

큐

배열을 이용한 큐 구현

- 큐 추출 함수

```
int pop() {
  if (front == rear) {
    printf("큐 언더플로우가 발생했습니다.\n");
    return -INF;
  }
  return queue[front++];
}
```

큐

배열을 이용한 큐 구현

- 큐 전체 출력 함수

```
void show() {
  printf("--- 큐의 앞 ---\n");
  for (int i = front; i < rear; i++) {
    printf("%d\n", queue[i]);
  }
  printf("--- 큐의 뒤 ---\n");
}
```

큐

배열을 이용한 큐 구현

- 완성된 큐 사용하기

```
int main(void) {
  push(7);
  push(5);
  push(4);
  pop();
  push(6);
  pop();
  show();
  system("pause");
  return 0;
}
```

큐

연결 리스트를 이용한 큐 구현

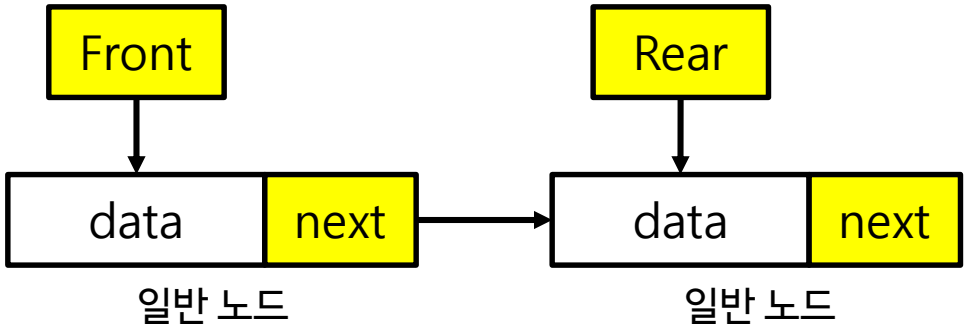
- 큐의 선언

```c
#include <stdio.h>
#include <stdlib.h>
#define INF 99999999

typedef struct {
  int data;
  struct Node *next;
} Node;

typedef struct {
  Node *front;
  Node *rear;
  int count;
} Queue;
```

큐

연결 리스트를 이용한 큐 구현

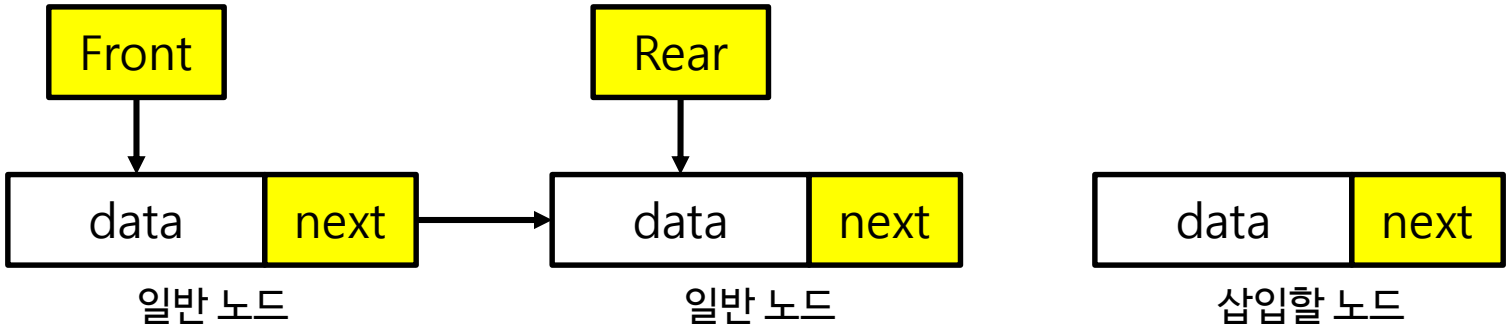
- 큐 삽입 과정 1



큐

연결 리스트를 이용한 큐 구현

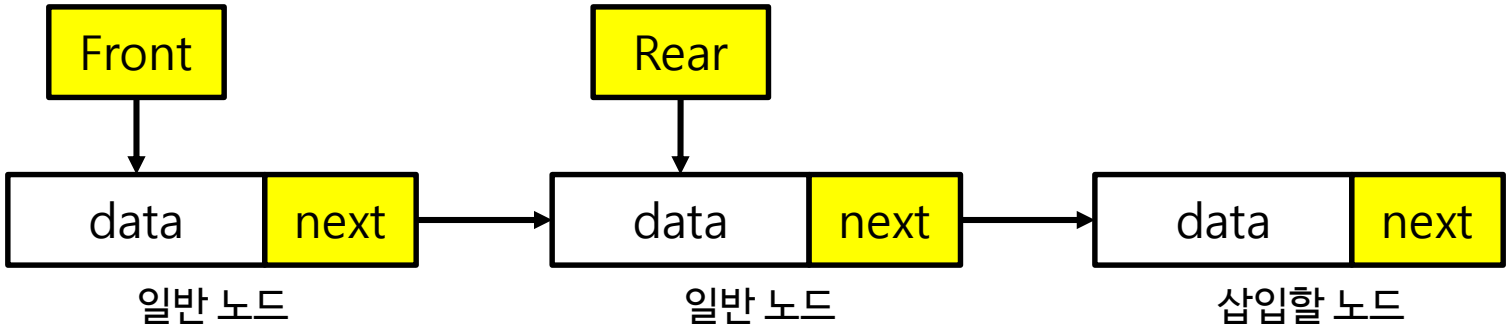
- 큐 삽입 과정 2



큐

연결 리스트를 이용한 큐 구현

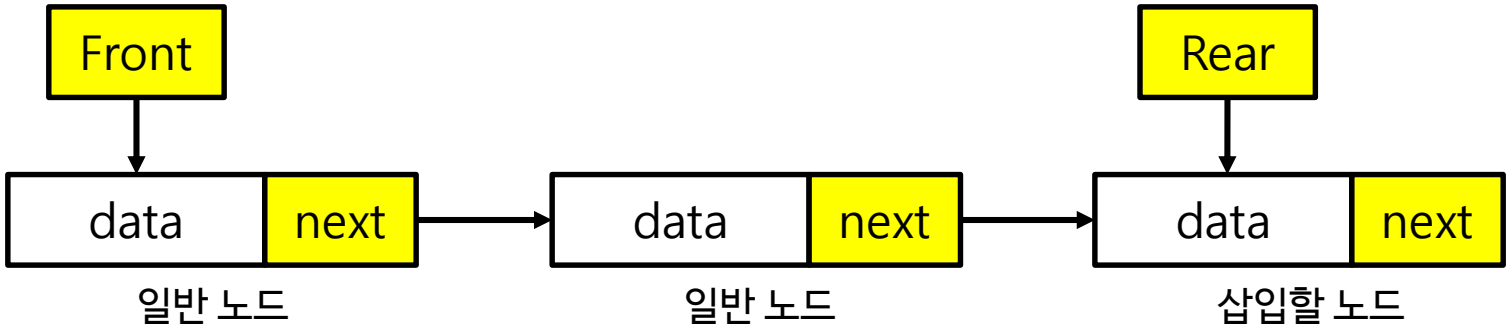
- 큐 삽입 과정 3



큐

연결 리스트를 이용한 큐 구현

- 큐 삽입 과정 4



큐

연결 리스트를 이용한 큐 구현

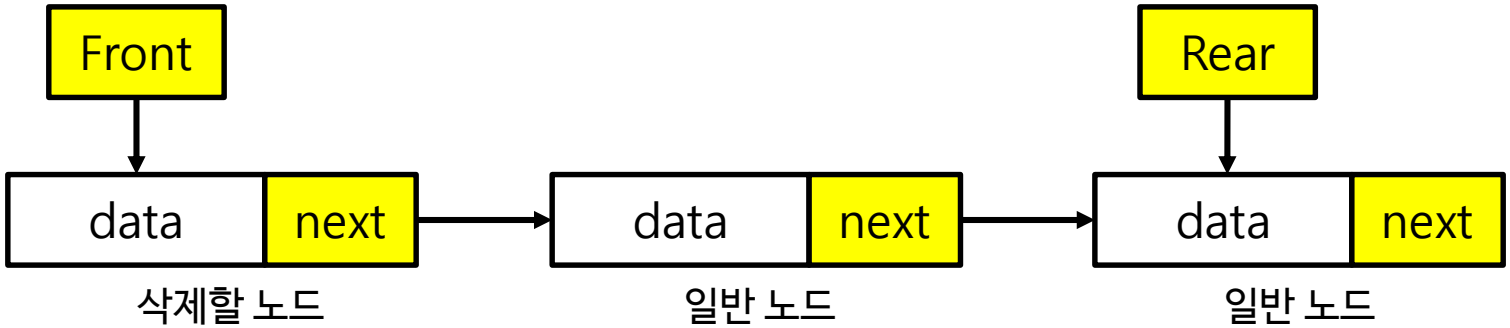
- 큐 삽입 함수

```
void push(Queue *queue, int data) {
  Node *node = (Node*)malloc(sizeof(Node));
  node->data = data;
  node->next = NULL;
  if (queue->count == 0) {
    queue->front = node;
  }
  else {
    queue->rear->next = node;
  }
  queue->rear = node;
  queue->count++;
}
```

큐

연결 리스트를 이용한 큐 구현

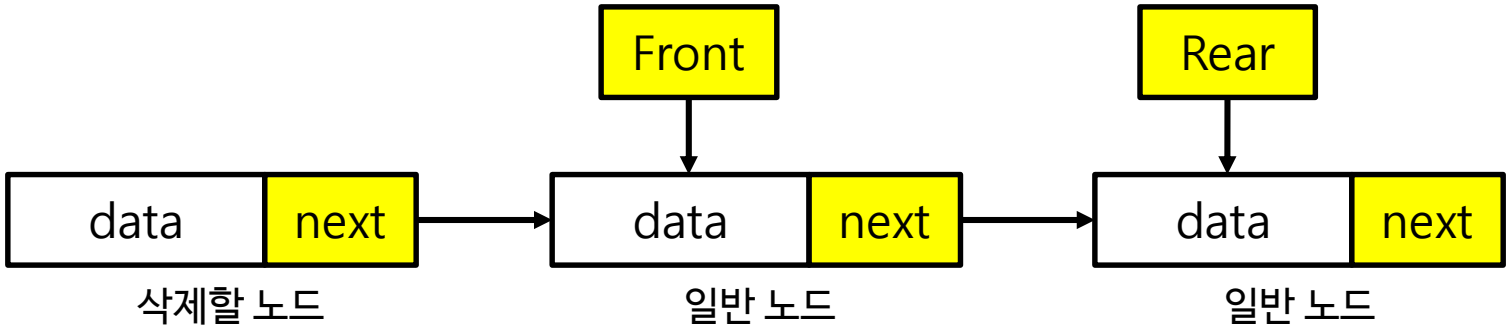
- 큐 추출 과정 1



큐

연결 리스트를 이용한 큐 구현

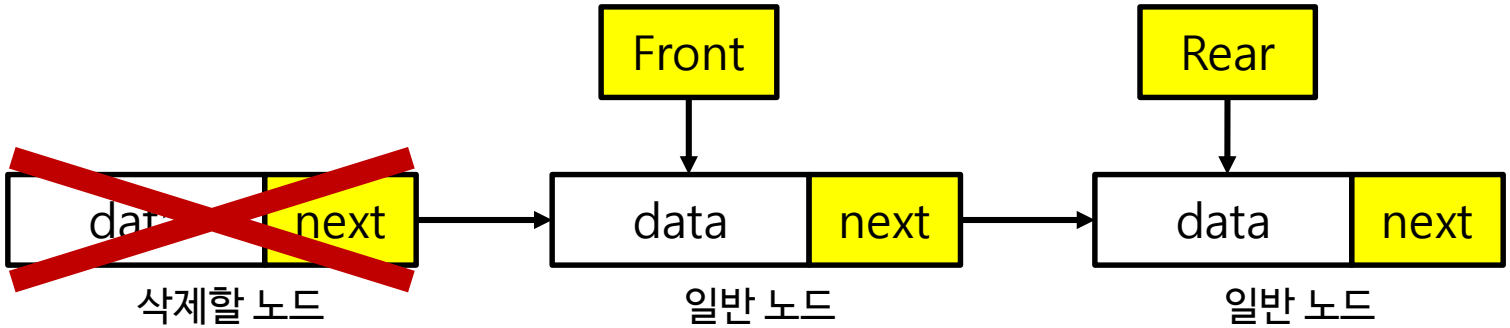
- 큐 추출 과정 2



큐

연결 리스트를 이용한 큐 구현

- 큐 추출 과정 3



큐

연결 리스트를 이용한 큐 구현

- 큐 추출 함수

```
int pop(Queue *queue) {
  if (queue->count == 0) {
    printf("큐 언더플로우가 발생했습니다.\n");
    return -INF;
  }
  Node *node = queue->front;
  int data = node->data;
  queue->front = node->next;
  free(node);
  queue->count--;
  return data;
}
```

큐

연결 리스트를 이용한 큐 구현

- 큐 전체 출력 함수

```
void show(Queue *queue) {
  Node *cur = queue->front;
  printf("--- 큐의 앞 ---\n");
  while (cur != NULL) {
    printf("%d\n", cur->data);
    cur = cur->next;
  }
  printf("--- 큐의 뒤 ---\n");
}
```

큐

연결 리스트를 이용한 큐 구현

- 완성된 큐 사용하기

```
int main(void) {
  Queue queue;
  queue.front = queue.rear = NULL;
  queue.count = 0;
  push(&queue, 7);
  push(&queue, 5);
  push(&queue, 4);
  pop(&queue);
  push(&queue, 6);
  pop(&queue);
  show(&queue);
  system("pause");
  return 0;
}
```

배운 내용 정리하기

큐

1) 큐는 배열 혹은 연결 리스트를 이용해서 구현할 수 있습니다.