Queue 자료 구조
구현 및 분석

작성자 : Lee Daero(skseofh@naver.com)

<자료 구조 Queue 구현 및 분석>

[목적]

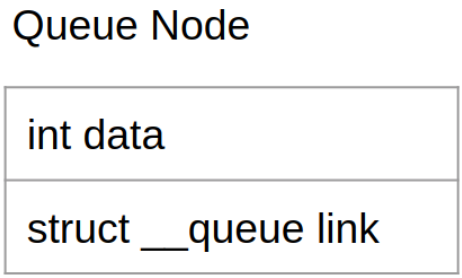
- queue 구조를 그림을 통해 분석한다.

[queue]

- First In First Out 의 특징을 가지는 자료구조
- 구현에서는 Linked List 형태로 구현한다.

(1) node 구조

```
typedef struct __queue
{
    int data;
    struct __queue *link;
} queue;
```



- data를 담는 int 형 변수
- 다음 node를 가리키는 struct __queue * 형 link 포인터

(2) 구현 함수

```
queue *get_node(void);
void enqueue(queue **head, int data);
void dequeue(queue **head, int data);
void print_queue(queue *head);
```

- get_node : 동적할당으로 queue 구조체를 할당 받아서 포인터를 반환
- enqueue : queue의 마지막에 data를 저장
- dequeue : queue에서 data에 해당하는 node를 제거
- print_queue : queue에 저장된 data를 출력

(3) main() 함수 동작 설명

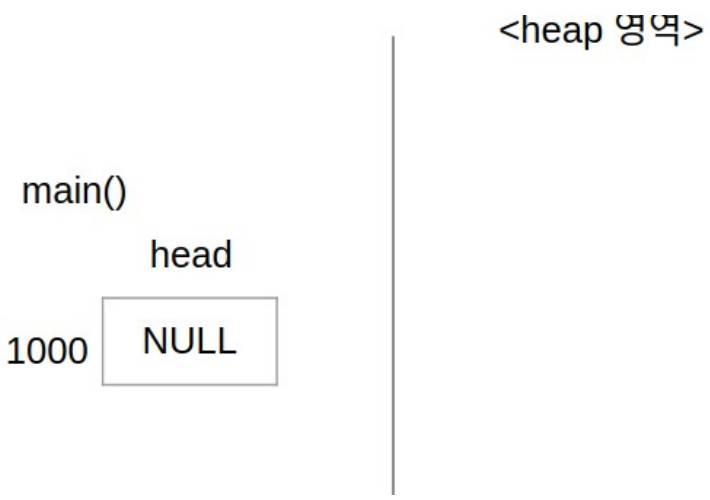
```c
int main(void)
{
    int i;

    queue *head = NULL;

    for(i = 0; i < 4; i++)
        enqueue(&head, (i +1) * 10);

    print_queue(head);

    dequeue(&head, 10);
    print_queue(head);

    dequeue(&head, 30);
    print_queue(head);

    dequeue(&head, 77);
    print_queue(head);

    return 0;
}
```

- 10, 20, 30, 40을 queue에 저장(enqueue)하고 queue를 출력한다.
- 10, 30, 77을 queue에서 제거하고 queue를 출력한다.
  queue에 data가 저장되어 있으면 제거하는 data를 출력한다.
  queue에 data가 저장되어 있지 않으면 에러 메세지를 출력한다.

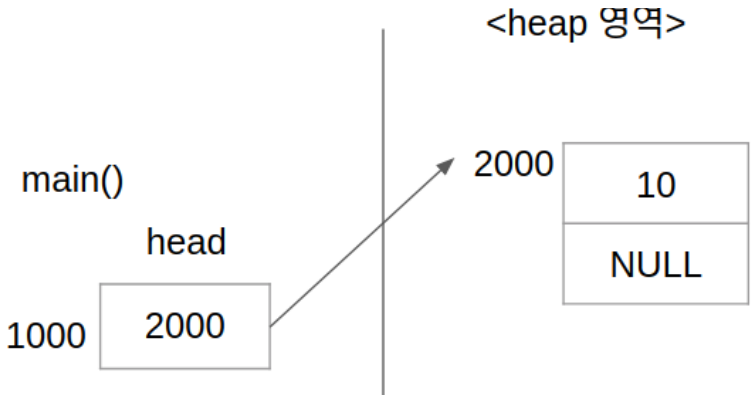
- 실행 화면

```
head->data = 10
head->data = 20
head->data = 30
head->data = 40
Now you delete 10
head->data = 20
head->data = 30
head->data = 40
Now you delete 30
head->data = 20
head->data = 40
There are no data that you delete
head->data = 20
head->data = 40
```

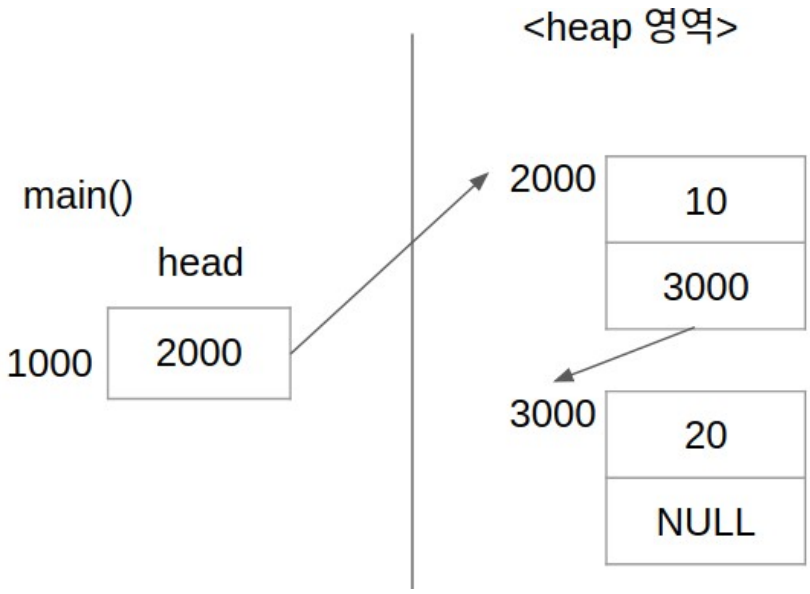
1) queue * head = NULL



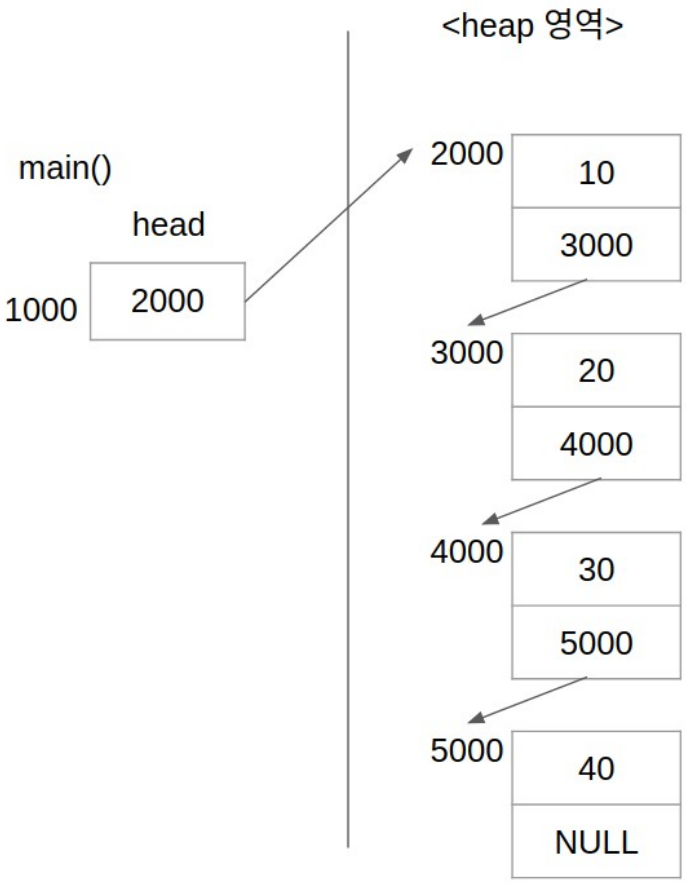
2) 10, 20, 30, 40 순서로 enqueue 진행
- enqueue() - 10



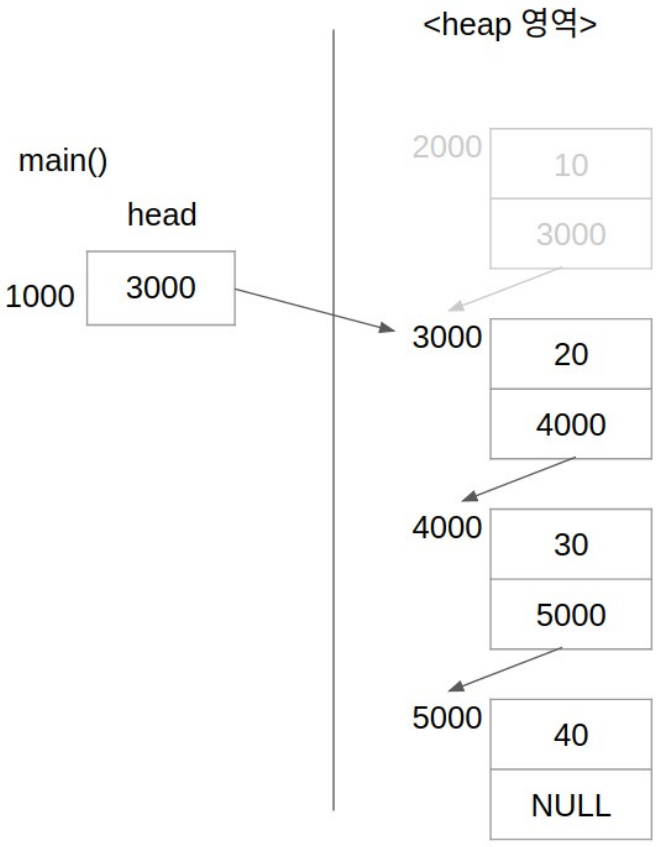
- enqueue() - 20



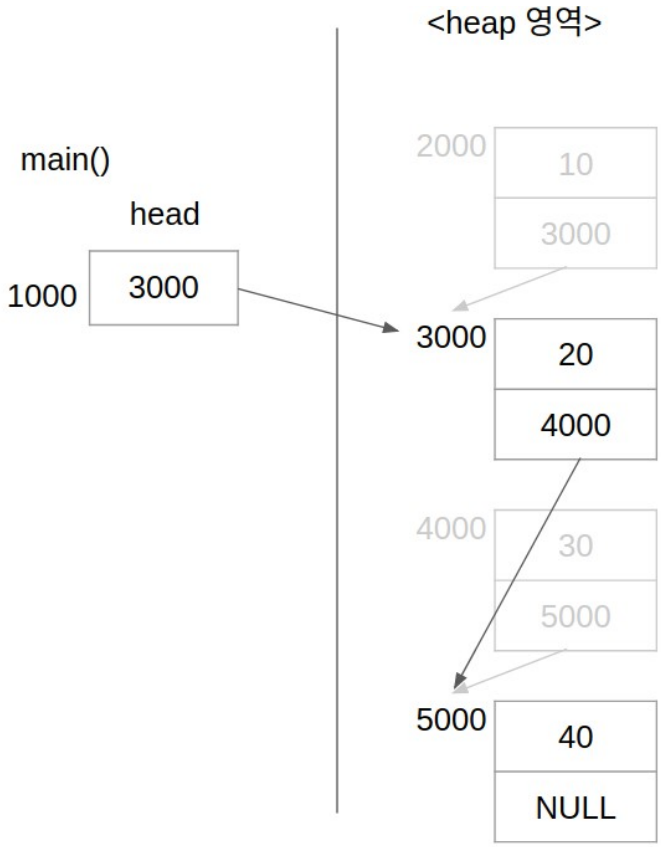
- enqueue() - 30, 40



3) dequeue() - data 10, 30
- dequeue : data 10



- dequeue : data 30



4) dequeue() - data 77

- queue에 data 77이 존재하지 않으므로 에러 메세지만 출력한다.

(4) 함수 분석

1) get_node : 동적할당으로 queue 구조체를 할당 받아서 포인터를 반환

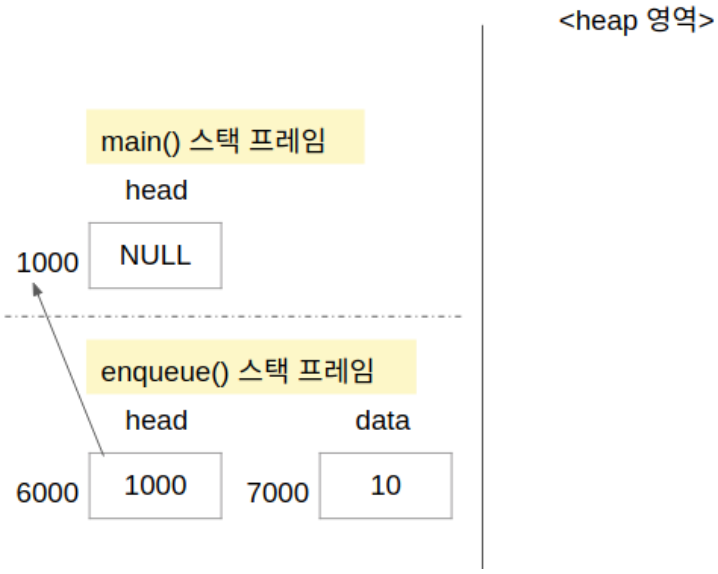
```
queue *get_node(void)
{
    queue *tmp;
    tmp = (queue *)malloc(sizeof(queue));
    tmp->link = NULL;
    return tmp;
}
```

- malloc 함수로 queue 구조체 만큼 queue * 형 tmp에 동적할당
- tmp의 다음 node를 가리키는 link를 NULL로 초기화
- tmp를 반환

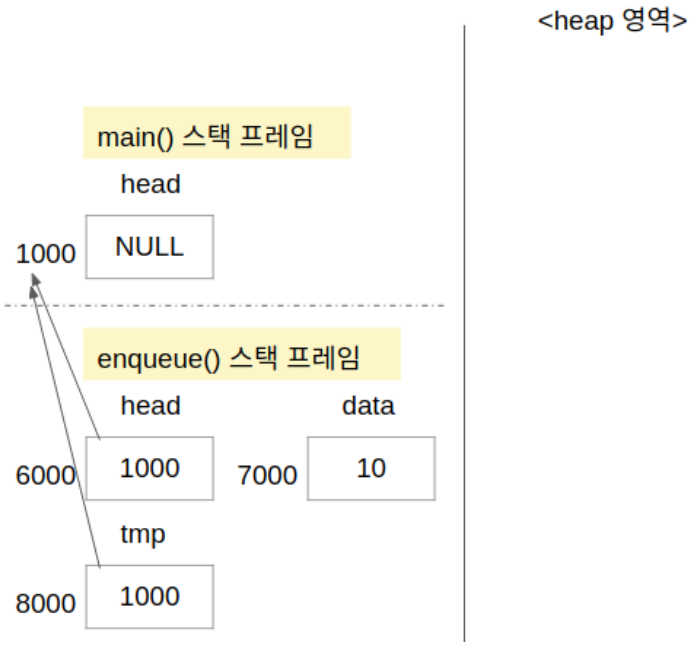
2) enqueue : queue의 마지막에 data를 저장

```
void enqueue(queue **head, int data)
{
    queue **tmp = head;

    while(*tmp)
        tmp = &(*tmp)->link;

    *tmp = get_node();
    (*tmp)->data = data;
}
```

- enqueue() 함수로 넘어 왔을 때 메모리 그림
  : enqueue() 스택 프레임에서 queue ** head이 main() 스택 프레임의 head 변수의 주소로 초기화되어 생성됨
  : int형인 변수가 data로 초기화 되어 생성

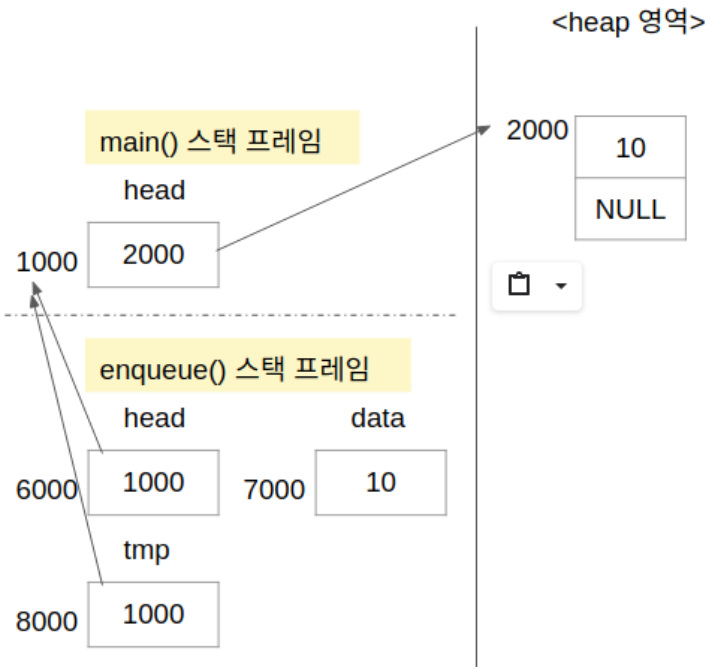


- queue ** tmp = head
 : queue ** 형의 tmp를 선언하고 head의 내용을 저장



- while 문
 : *tmp가 NULL일 때까지 queue를 타고 내려가서 node를 추가할 위치를 tmp에 저장함

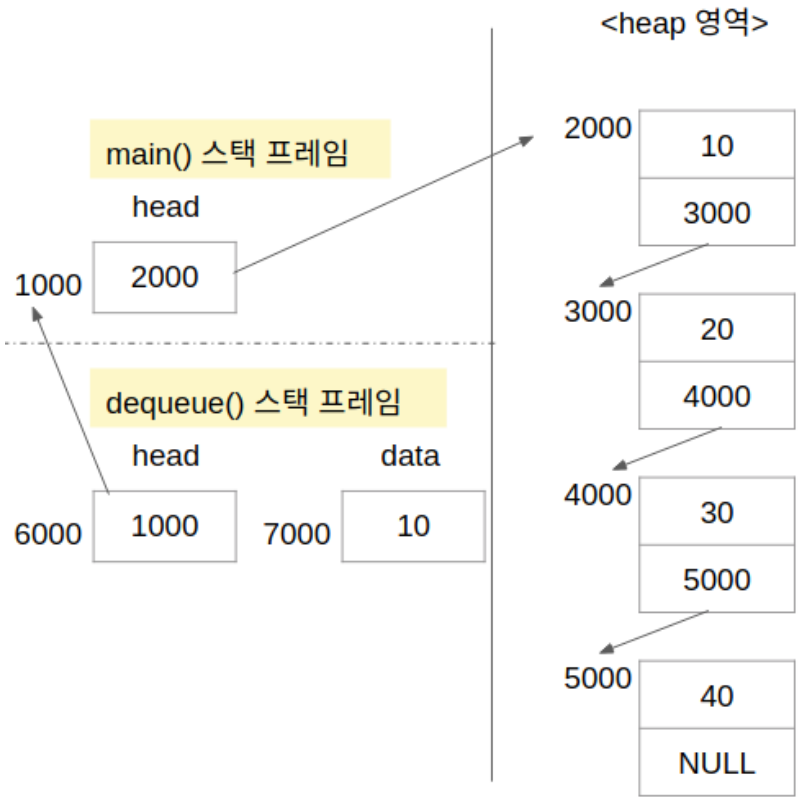
- *tmp = get_node()
- (*tmp)→data = data;
 : queue 노드를 동적할당 한 후 , data를 저장



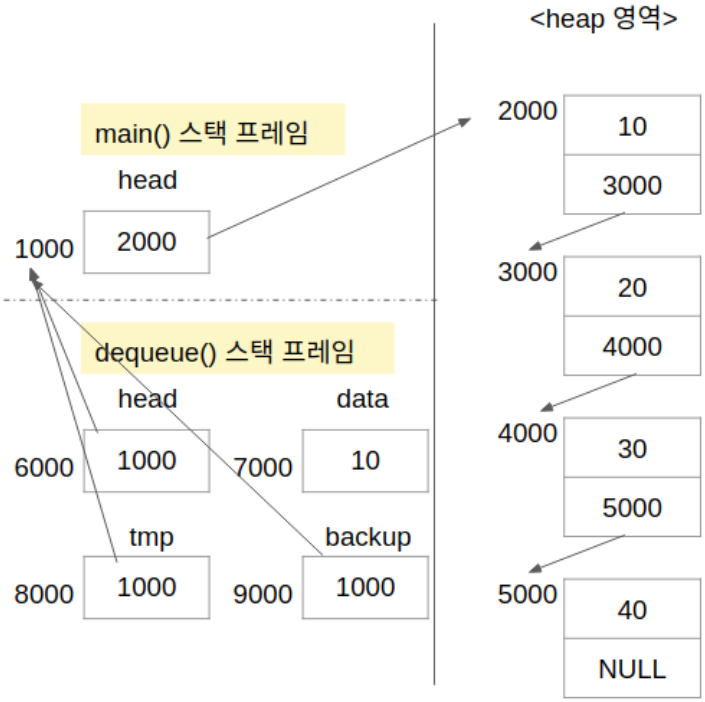
3) dequeue : queue에서 data에 해당하는 node를 제거

```
void dequeue(queue **head, int data)
{
    queue **tmp = head;
    queue **backup = tmp;

    if(*tmp == NULL)
    {
finish:
        printf("There are no data that you delete\n");
        return;
    }

    while((*tmp)->data != data)
    {
        if((*tmp)->link)
        {
            backup = &(*tmp)->link;
            tmp = &(*tmp)->link;
        }
        else
            goto finish;
    }

    printf("Now you delete %d\n", data);
    free(*tmp);
    *backup = (*tmp)->link;
}
```

- 10, 20, 30, 40이 queue에 저장되어 있는 상태로 가정
  #data 10을 인자로 하여 dequeue() 함수로 넘어왔을 때 메모리 그림

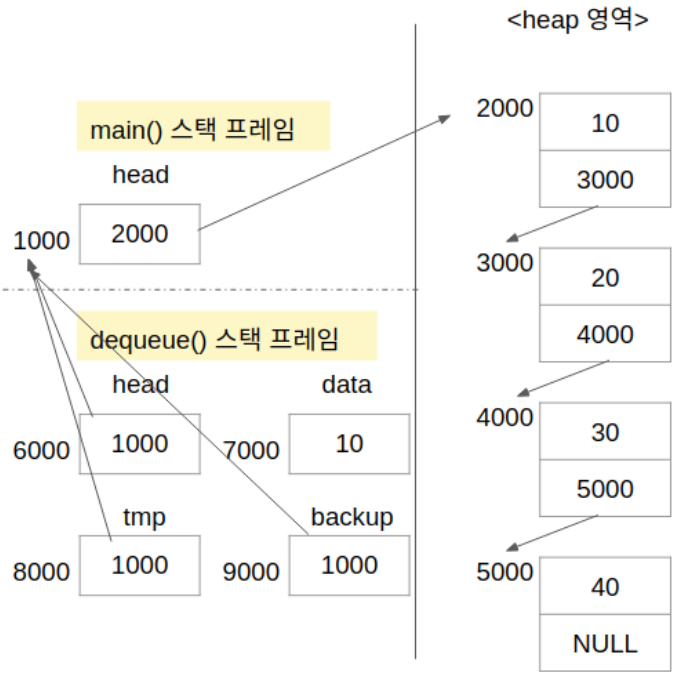


- queue ** tmp = head;
  queue **backup = tmp;
 : tmp – queue 노드를 타고 dequeue할 data를 찾을 때 쓰이는 queue ** 형 변수
  backup – 지울 node는 free() 함수를 이용해서 동적할당 메모리를 해제하는데, 지워진 node의 앞과 뒤의 node를 연결시켜 주기 위해 지울 node의 link변수의 주소를 저장하는 용도

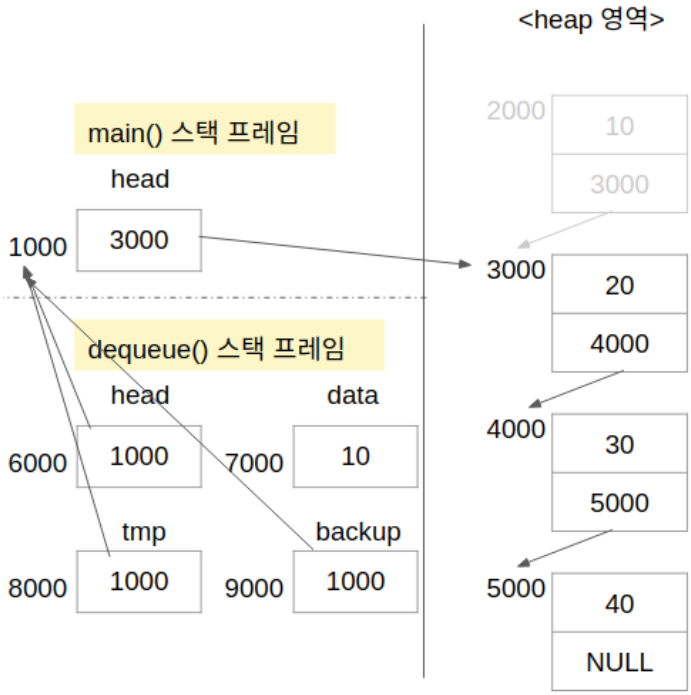


- if 문
 : queue의 시작 주소를 참조해서 가리키고 있는 tmp,
  *tmp가 NULL이면, 즉, queue에 저장된 data가 없으면 에러 메세지를 출력하고 return

- while 문 : dequeue() 할 data를 찾을 때까지 동작
- if 문 : (*tmp) 노드에 연결된 다음 노드가 있으면 backup, tmp에 다음 노드의 link의 주소를 저장
  없으면, goto 문을 이용, 에러 메세지 출력 후 return
 ## 여기서는 지울 data가 10이므로 tmp, backup 값의 변화가 없음

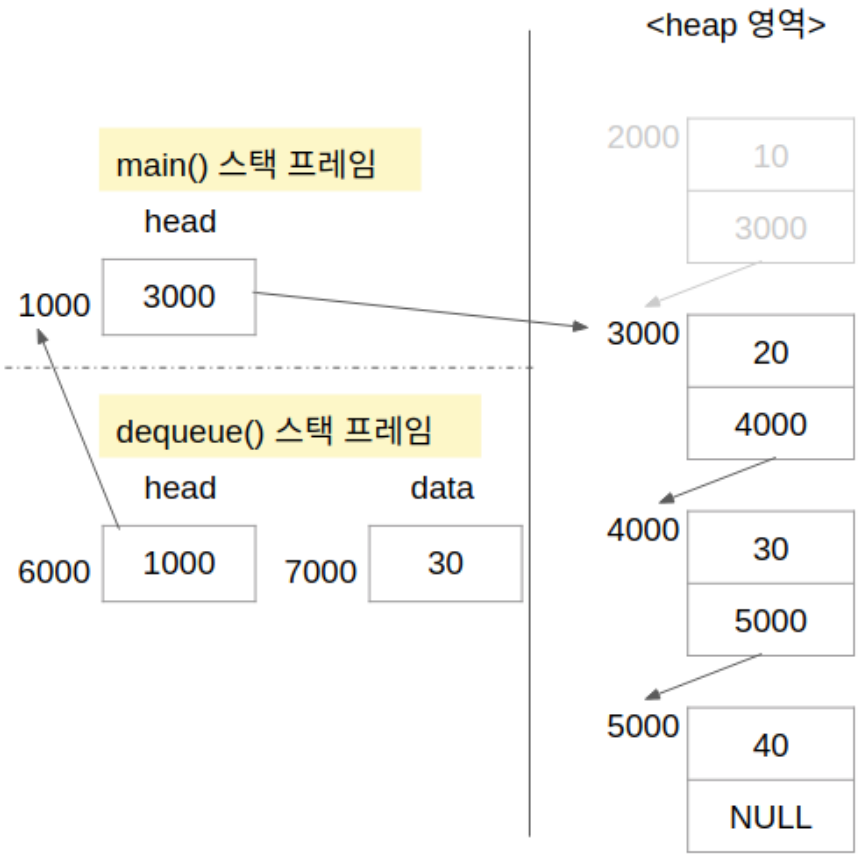


- printf 문 : 지울 데이터 출력
- free(*tmp)
  : 현재 tmp가 주소 참조해서 가리키고 있는 node를 free 함
- *backup = (*tmp)→link
  : 지운 node의 다음 node를 연결하기 위해 backup이 주소 참조하는 곳에 다음 node의 주소를 저장

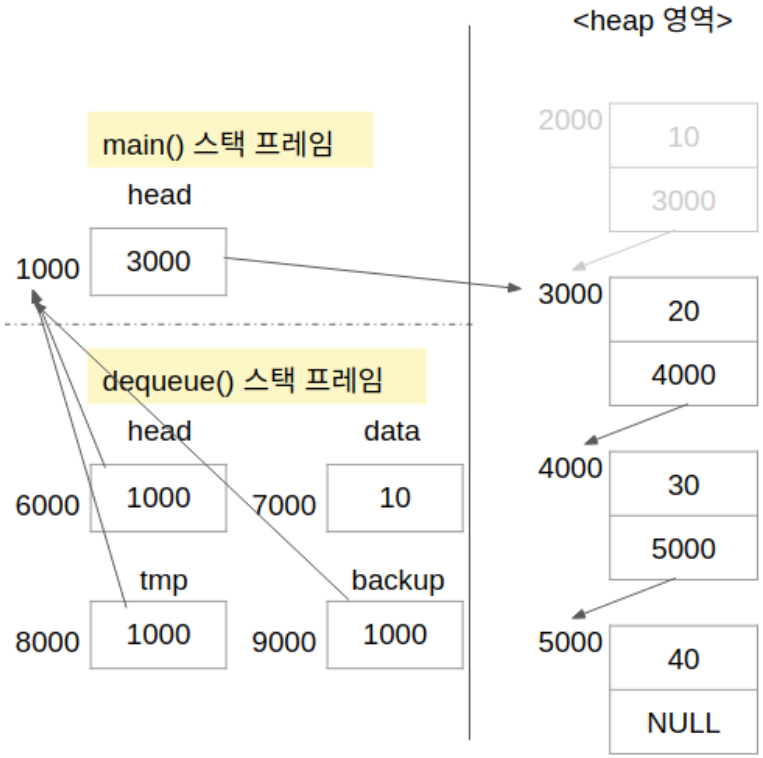


free(*tmp) 후 다시 (*tmp)를 참조해서 에러가 나지 않는 이유는?
→ 운영체제가 자원이 사용되지 않는 것을 정리할 때 그때그때 하지 않고 한번에 몰아서 하기 때문

data 30을 인자로 dequeue() 함수로 넘어왔을 때



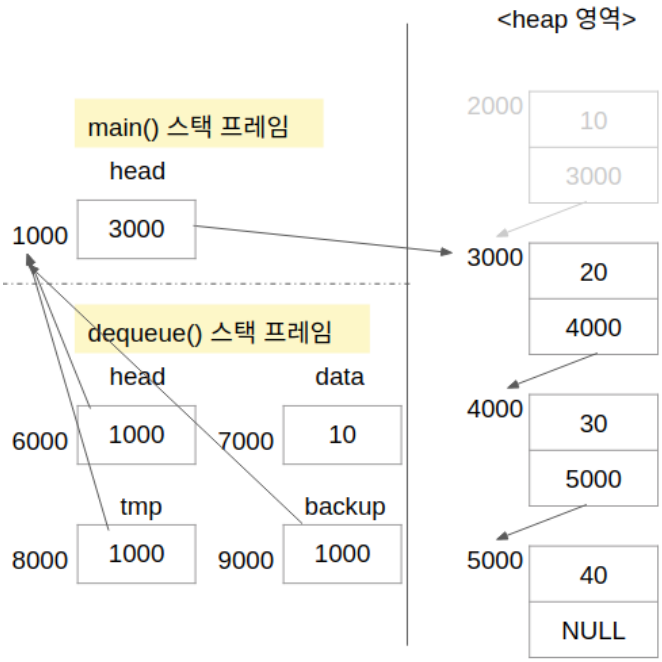
- queue ** tmp = head;
  queue **backup = tmp;



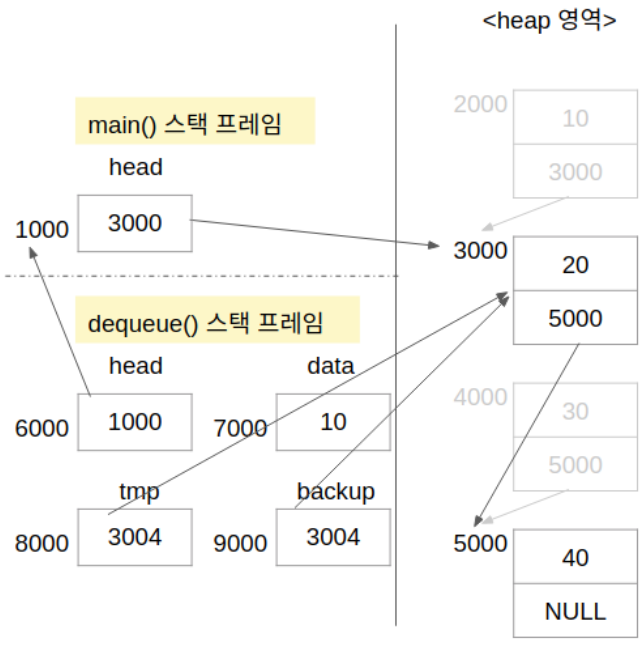
- if 문 : (*tmp)가 NULL 인지 확인
→ NULL이 아니므로 실행 X

- while문 + if 문
: data가 30이고 다음 node가 있을 때까지 link를 타고 내려감

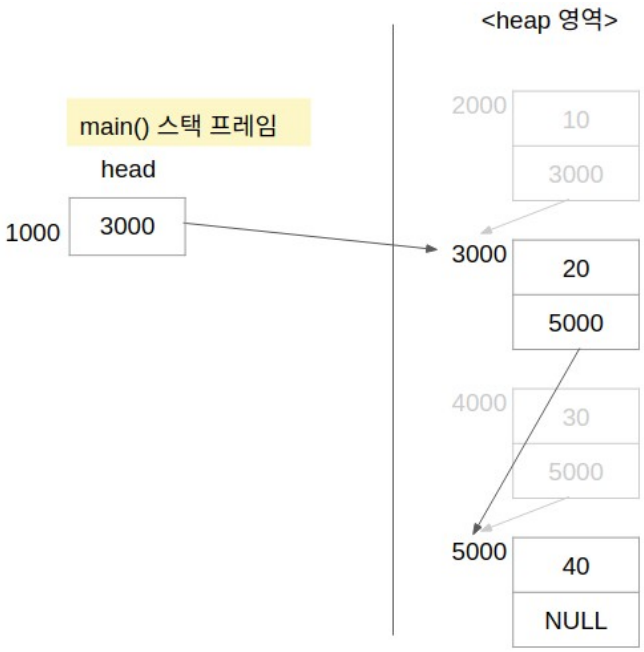
→ 첫번째 while 문을 만났을 때는 data가 30이 아니므로 link를 타고 내려감
→ 두번째 while 문을 만났을 때 data가 30이므로 if 문을 실행
, (*tmp)->link가 존재하므로 backup, tmp를 현재 node의 link 변수의 주소로 변경



- free(*tmp)
*backup = (*tmp)→link;
: data 30이 저장되어 있는 node를 free하고 (*backup)을 지울 node의 다음 node로 연결한다.



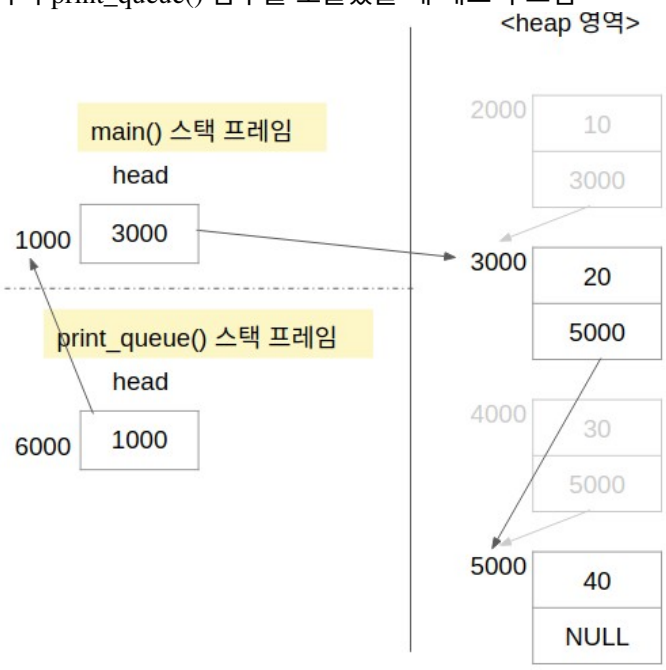
- dequeue() 함수가 끝나고 main()으로 돌아갔을 때



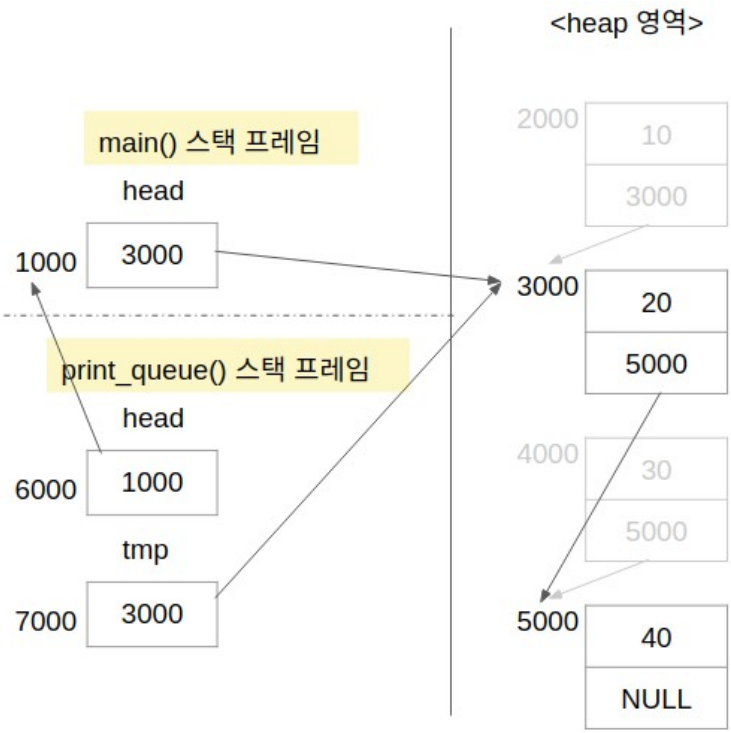
4) print_queue : queue에 저장된 data를 출력

```
void print_queue(queue *head)
{
    queue *tmp = tmp;
    while(tmp)
    {
        printf("head->data = %d\n", tmp->data);
        tmp = tmp->link;
    }
}
```

- 10, 30을 dequeue() 하고 나서 print_queue() 함수를 호출했을 때 메모리 그림



- queue * tmp = head



- while 문
  : tmp가 NULL이 아닐 때까지 link를 타고 내려가면서 data를 출력