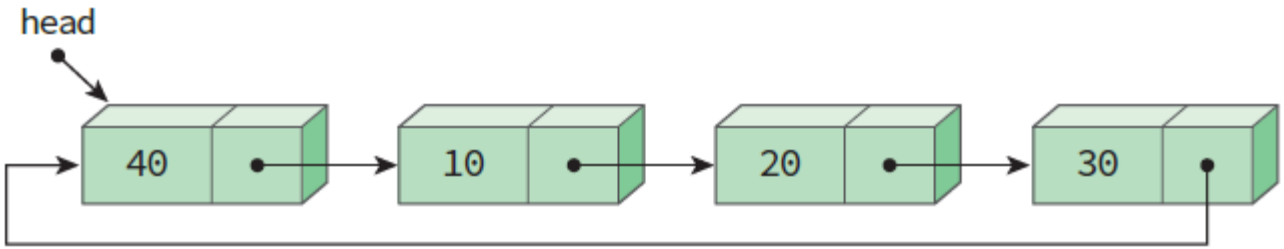
기장 연결 리스트 II

원형 연결리스트, 이중 연결리스트, 연결리스트 스택, 큐 구현

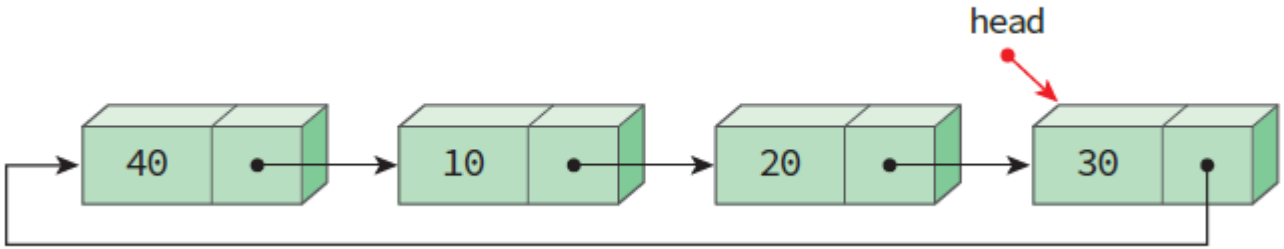
7.1 원형 연결 리스트

- 마지막 노드의 링크가 첫 번째 노드를 가리키는 리스트
- 한 노드에서 다른 모든 노드로의 접근이 가능

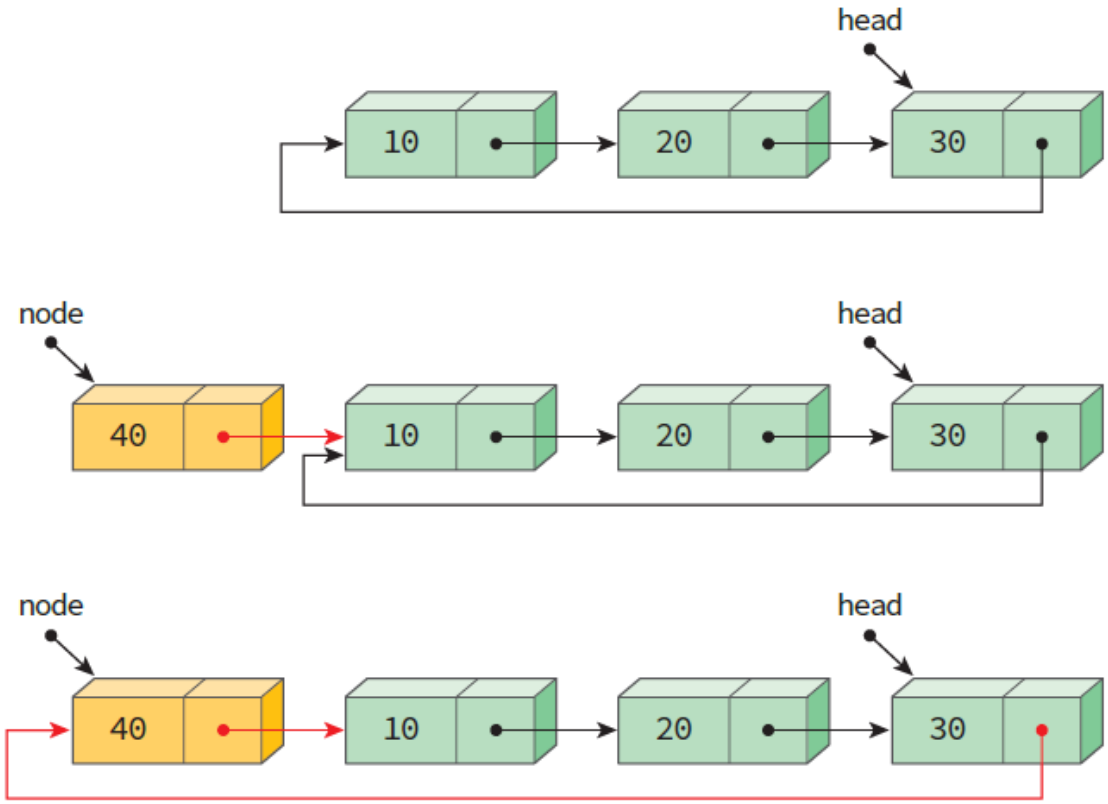


원형 연결 리스트 변형

- 보통 헤드포인터가 마지막 노드를 가리키게끔 구성
  - 리스트의 첫번째 노드는 head->link 로 지정
  - 장점: 리스트의 처음이나 마지막에 노드를 삽입하는 연산이 단순 연결 리스트에 비하여 용이



원형 연결 리스트의 처음에 삽입

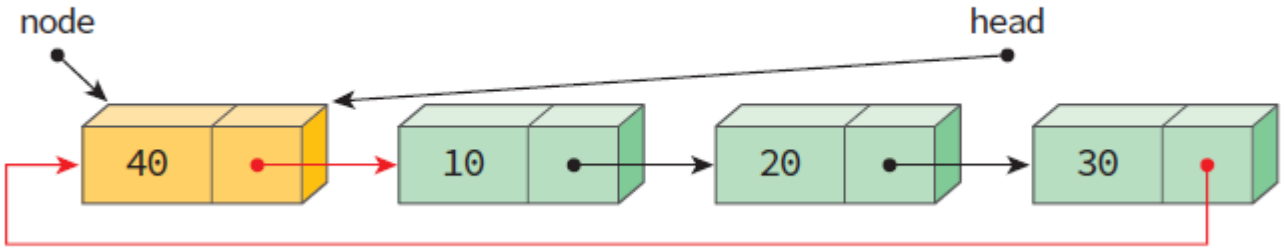


원형 연결리스트의 처음에 삽입


```
24  ListNode* insert_first(ListNode* head, element data)
25  {
26      ListNode *node = (ListNode *)malloc(sizeof(ListNode));
27      node->data = data;
28      if (head == NULL)
29      {
30          head = node;
31          node->link = head;
32      }
33      else
34      {
35          node->link = head->link;     // (1)
36          head->link = node;           // (2)
37      }
38      return head;                     // 변경된 헤드 포인터를 반환한다.
39  }
```

원형 연결리스트의 끝에 삽입


```
41  ListNode* insert_last(ListNode* head, element data)
42  {
43      ListNode *node = (ListNode *)malloc(sizeof(ListNode));
44      node->data = data;
45      if (head == NULL)
46      {
47          head = node;
48          node->link = head;
49      }
50      else
51      {
52          node->link = head->link;     // (1)
53          head->link = node;           // (2)
54          head = node;                 // (3)
55      }
56      return head;          // 변경된 헤드 포인터를 반환한다.
57  }
```



원형 연결리스트 테스트 프로그램: cir_list.c (1/2)


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

typedef int element;
typedef struct ListNode {    // 노드 타입
    element data;
    struct ListNode *link;
} ListNode;

// 리스트의 항목 출력:
void print_list(ListNode* head)
{
    ListNode* p;

    if (head == NULL) return;
    p = head->link;
    do {
        printf("%d->", p->data);
        p = p->link;
    } while (p != head);
    printf("%d->", p->data);     // 마지막 노드 출력
}
```

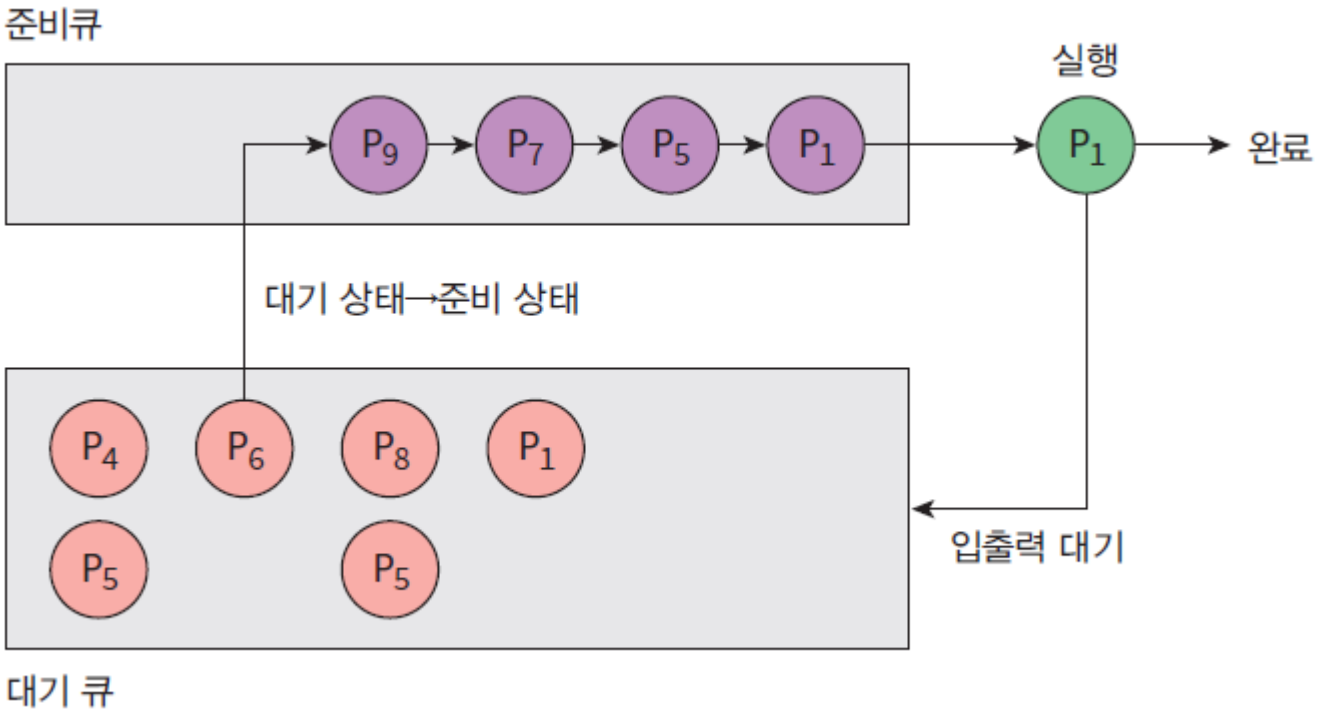
원형 연결리스트 테스트 프로그램: cir_list.c (2/2)


```
59  // 원형 연결 리스트 테스트 프로그램
60  int main(void)
61  {
62      ListNode *head = NULL;
63
64      // list = 10->20->30->40
65      head = insert_last(head, 20);
66      head = insert_last(head, 30);
67      head = insert_last(head, 40);
68      head = insert_first(head, 10);
69      print_list(head);
70      return 0;
71  }
```

```
10->20->30->40->
--------------------------------
Process exited after 0.02418 seconds with return value 0
계속하려면 아무 키나 누르십시오 . . .
```

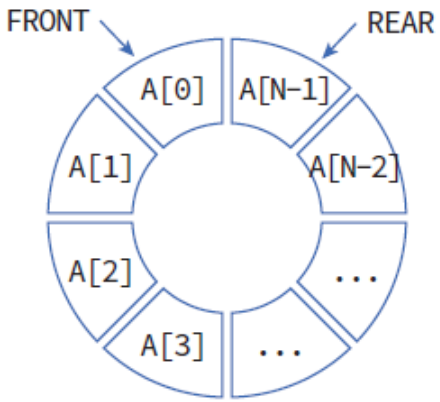
7.2 원형 연결 리스트의 응용 (1/2)

- Process Scheduling

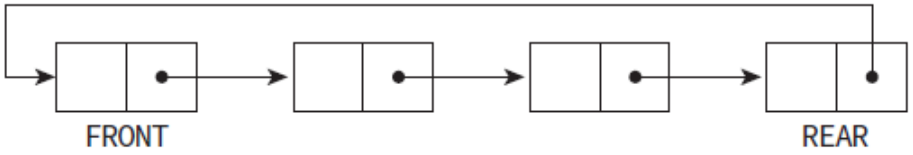


원형 연결 리스트의 응용 (2/2)

- 멀티플레이어 게임
- 원형 큐



배열을 이용한 원형큐



연결 리스트를 이용한 원형큐

- head를 REAR로 head->link로 FRONT를 포인팅
- insert_last(head, data)로 enqueque(q, element) 구현
- delete_first(head)로 dequeque(q)를 구현

멀티 플레이어 게임 구현


```
현재 차례=KIM
현재 차례=CHOI
현재 차례=PARK
현재 차례=KIM
현재 차례=CHOI
현재 차례=PARK
현재 차례=KIM
현재 차례=CHOI
현재 차례=PARK
현재 차례=KIM
```

멀티 플레이어 게임: multigame.c (1/2)


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

typedef char element[100];
typedef struct ListNode {    // 노드 타입
    element data;
    struct ListNode *link;
} ListNode;

ListNode *insert_first(ListNode *head, element data)
{
    ListNode *node = (ListNode *)malloc(sizeof(ListNode));
    strcpy(node->data, data);
    if (head == NULL) {
        head = node;
        node->link = head;
    }
    else {
        node->link = head->link;    // (1)
        head->link = node;          // (2)
    }
    return head;
}
```

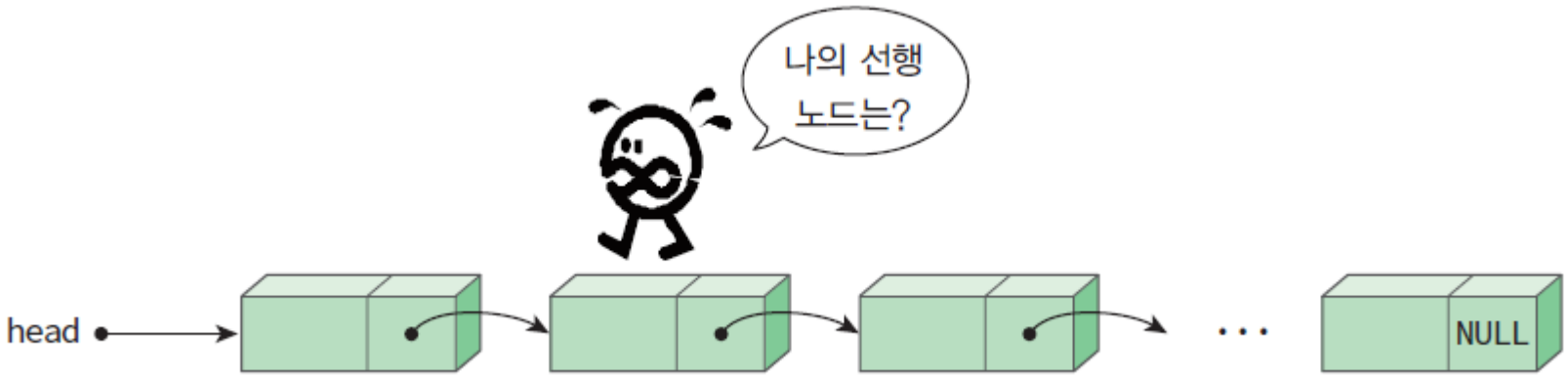
멀티 플레이어 게임: multigame.c (1/2)


```
30  // 원형 연결 리스트 테스트 프로그램
31  int main(void)
32  {
33      int i;
34      ListNode *head = NULL, *p;
35
36      head = insert_first(head, "KIM");
37      head = insert_first(head, "PARK");
38      head = insert_first(head, "CHOI");
39
40      p = head->link;
41      for (i=0; i< 10; i++) {
42          printf("현재 차례=%s \n", p->data);
43          p = p->link;
44      }
45      return 0;
46  }
```

```
현재 차례=CHOI
현재 차례=PARK
현재 차례=KIM
현재 차례=CHOI
현재 차례=PARK
현재 차례=KIM
현재 차례=CHOI
현재 차례=PARK
현재 차례=KIM
현재 차례=CHOI

--------------------------------
Process exited after 0.01785 seconds with return value 0
계속하려면 아무 키나 누르십시오 . . .
```

7.3 이중 연결리스트

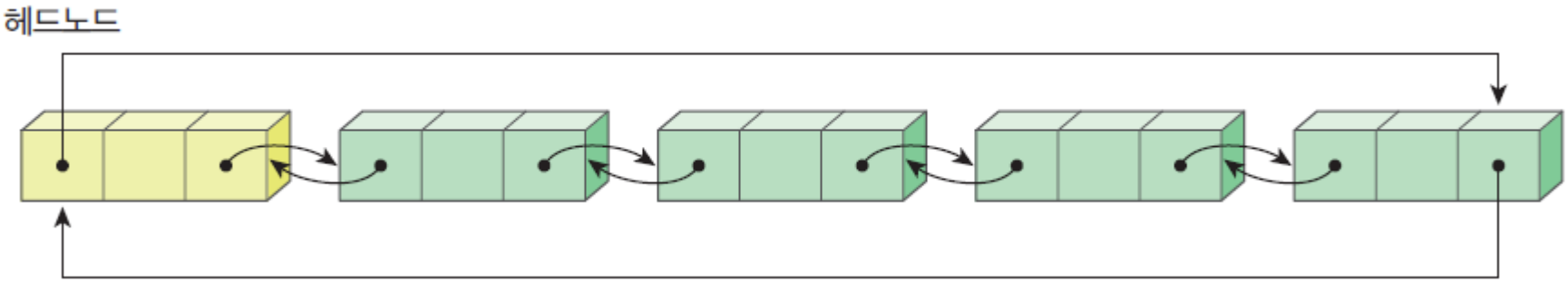
- 단순 연결 리스트의 문제점: 선행 노드를 찾기가 힘들다



이중 연결리스트 구조

- 이중 연결리스트: 하나의 노드가 선행 노드와 후속 노드에 대한 두 개의 링크를 가지는 리스트
- 단점: 공간을 많이 차지하고 코드가 복잡

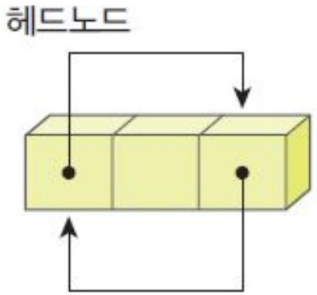
- 구현 예: 이중 연결리스트 + 원형 연결리스트



헤드 노드

- 헤드 노드(head node): 데이터를 가지지 않고 단지 삽입, 삭제 코드를 간단하게 할 목적으로 만들어진 노드
  - 일반 노드와 동일한 구조지만 데이터를 포함하지 않음
    - 헤드 포인터와의 구별 필요
  - 공백상태에서는 헤드 노드만 존재

[그림 7-7] 이중 연결 리스트에서의 노드의 구조



[그림 7-8] 공백상태

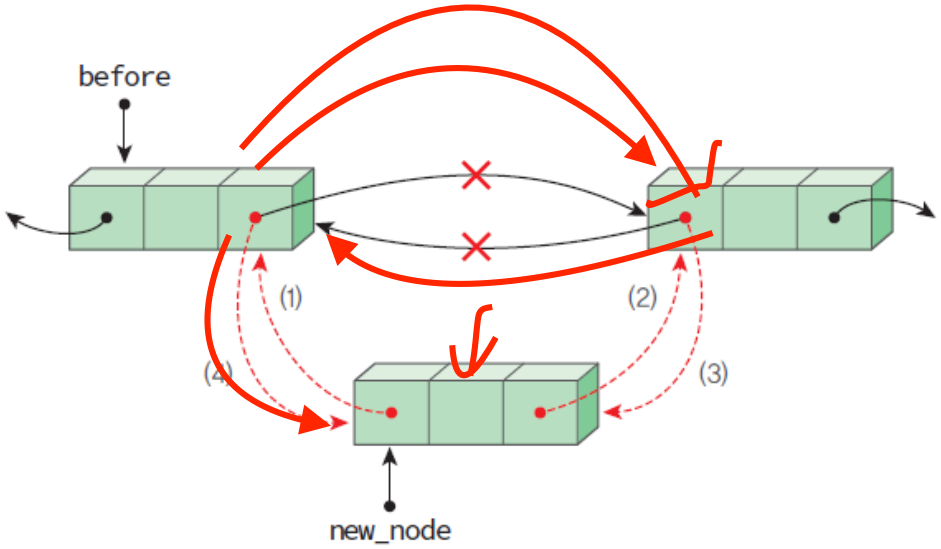
이중 연결리스트 노드의 정의

- 이중연결리스트에서의 노드의 구조

```
4   typedef int element;
5   typedef struct DListNode {  // 이중연결 노드 타입
6       element data;
7       struct DListNode* llink;
8       struct DListNode* rlink;
9   } DListNode;
```

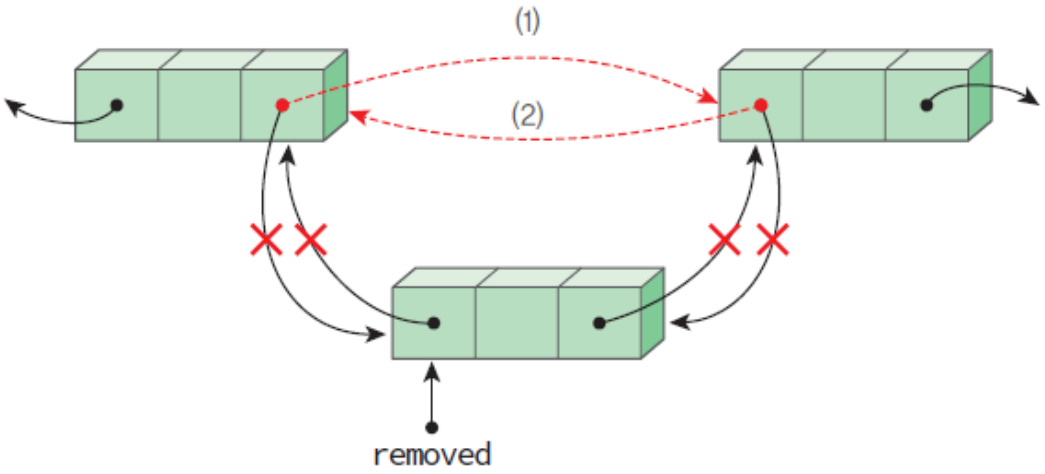
이중 연결리스트 삽입 연산


```
28  // 새로운 데이터를 노드 before의 오른쪽에 삽입한다.
29  void dinsert(DListNode *before, element data)
30  {
31      DListNode *newnode = (DListNode *)malloc(sizeof(DListNode));
32      newnode->data= data;
33      newnode->llink = before;
34      newnode->rlink = before->rlink;
35      before->rlink->llink = newnode;
36      before->rlink = newnode;
37  }
```



이중 연결리스트 삭제 연산


```
39  // 노드 removed를 삭제한다.
40  void ddelete(DListNode* head, DListNode* removed)
41  {
42      if (removed == head) return;
43      removed->llink->rlink = removed->rlink;
44      removed->rlink->llink = removed->llink;
45      free(removed);
46  }
```



이중 연결리스트 테스트 프로그램: dlinkedlist.c (1/3)


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

typedef int element;
typedef struct DListNode {  // 이중 연결 노드 타입
    element data;
    struct DListNode* llink;
    struct DListNode* rlink;
} DListNode;

// 이중 연결 리스트를 초기화
void init(DListNode* phead)
{
    phead->llink = phead;
    phead->rlink = phead;
}

// 이중 연결 리스트의 노드를 출력
void print_dlist(DListNode* phead)
{
    DListNode* p;
    for (p = phead->rlink; p != phead; p = p->rlink) {
        printf("<-| |%d| |-> ", p->data);
    }
    printf("\n");
}
```

이중 연결리스트 테스트 프로그램: dlinkedlist.c (2/3)

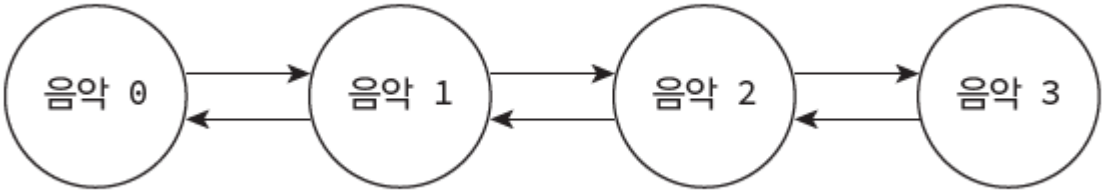

```
28  // 새로운 데이터를 노드 before의 오른쪽에 삽입한다.
29  void dinsert(DListNode *before, element data)
30  {
31      DListNode *newnode = (DListNode *)malloc(sizeof(DListNode));
32      newnode->data= data;
33      newnode->llink = before;
34      newnode->rlink = before->rlink;
35      before->rlink->llink = newnode;
36      before->rlink = newnode;
37  }
38
39  // 노드 removed를 삭제한다.
40  void ddelete(DListNode* head, DListNode* removed)
41  {
42      if (removed == head) return;
43      removed->llink->rlink = removed->rlink;
44      removed->rlink->llink = removed->llink;
45      free(removed);
46  }
```

이중 연결리스트 테스트 프로그램: dlinkedlist.c (3/3)


```c
// 이중 연결 리스트 테스트 프로그램
int main(void)
{
    int i;
    DListNode* head = (DListNode *)malloc(sizeof(DListNode));
    init(head);
    printf("추가 단계 \n");
    for (i = 0; i < 5; i++) {
        // 헤드 노드의 오른쪽에 삽입
        dinsert(head, i);
        print_dlist(head);
    }
    printf("\n삭제 단계 \n");
    for (i = 0; i < 5; i++) {
        print_dlist(head);
        ddelete(head, head->rlink);
    }
    free(head);
    return 0;
}
```

```
추가 단계
<-| |0| |->
<-| |1| |-> <-| |0| |->
<-| |2| |-> <-| |1| |-> <-| |0| |->
<-| |3| |-> <-| |2| |-> <-| |1| |-> <-| |0| |->
<-| |4| |-> <-| |3| |-> <-| |2| |-> <-| |1| |-> <-| |0| |->

삭제 단계
<-| |4| |-> <-| |3| |-> <-| |2| |-> <-| |1| |-> <-| |0| |->
<-| |3| |-> <-| |2| |-> <-| |1| |-> <-| |0| |->
<-| |2| |-> <-| |1| |-> <-| |0| |->
<-| |1| |-> <-| |0| |->
<-| |0| |->
```

7.4 예제 mp3 재생 프로그램 만들기




```
<-| #Fernando# |-> <-| Dancing Queen |-> <-| Mamamia |->

명령어를 입력하시오(<, >, q): >
<-| Fernando |-> <-| #Dancing Queen# |-> <-| Mamamia |->

명령어를 입력하시오(<, >, q): >
<-| Fernando |-> <-| Dancing Queen |-> <-| #Mamamia# |->

명령어를 입력하시오(<, >, q): <
<-| Fernando |-> <-| #Dancing Queen# |-> <-| Mamamia |->

명령어를 입력하시오(<, >, q):
```

mp3 재생 프로그램: mp3_play.c (1/2)


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

typedef char element[100];
typedef struct DListNode {  // 이중 연결 노드 타입
    element data;
    struct DListNode* llink;
    struct DListNode* rlink;
} DListNode;

DListNode* current;

// 이중 연결 리스트를 초기화
void init(DListNode* phead)
{
    phead->llink = phead;
    phead->rlink = phead;
}

// 이중 연결 리스트의 노드를 출력
void print_dlist(DListNode* phead)
{
    DListNode* p;
    for (p = phead->rlink; p != phead; p = p->rlink) {
        if (p == current)
            printf("<-| #%s# |-> ", p->data);
        else
            printf("<-| %s |-> ", p->data);
    }
    printf("\n");
}
```

mp3 재생 프로그램: mp3_play.c (2/2)

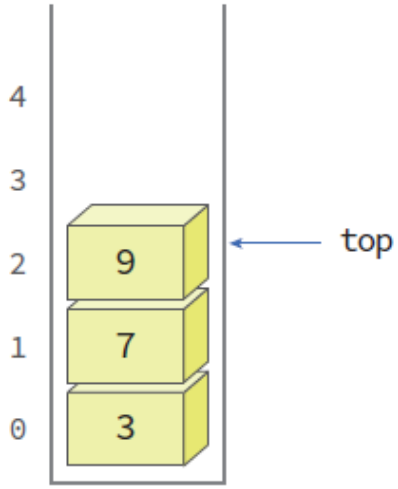

```c
// 이중 연결 리스트 테스트 프로그램
int main(void)
{
    char ch;
    DListNode* head = (DListNode *)malloc(sizeof(DListNode));
    init(head);

    dinsert(head, "Mamamia");
    dinsert(head, "Dancing Queen");
    dinsert(head, "Fernando");

    current = head->rlink;
    print_dlist(head);

    do {
        printf("\n명령어를 입력하시오 (<, >, q): ");
        ch = getchar();
        if (ch == '<') {
            current = current->llink;
            if (current == head)
                current = current->llink;
        }
        else if (ch == '>') {
            current = current->rlink;
            if (current == head)
                current = current->rlink;
        }
        print_dlist(head);
        getchar();
    } while (ch != 'q');
    // 동적 메모리 해제 코드를 여기에
}
```

mp3 재생 프로그램: 실행 예

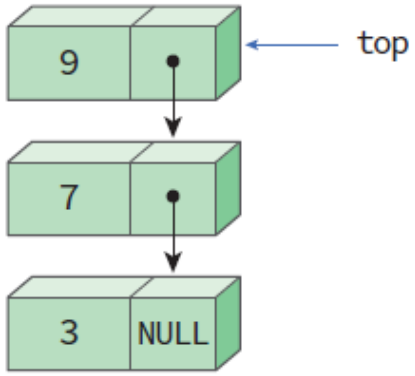

```
<-| #Fernando# |-> <-| Dancing Queen |-> <-| Mamamia |->

명령어를 입력하시오(<, >, q): >
<-| Fernando |-> <-| #Dancing Queen# |-> <-| Mamamia |->

명령어를 입력하시오(<, >, q): >
<-| Fernando |-> <-| Dancing Queen |-> <-| #Mamamia# |->

명령어를 입력하시오(<, >, q): <
<-| Fernando |-> <-| #Dancing Queen# |-> <-| Mamamia |->

명령어를 입력하시오(<, >, q): <
<-| #Fernando# |-> <-| Dancing Queen |-> <-| Mamamia |->

명령어를 입력하시오(<, >, q): q
<-| #Fernando# |-> <-| Dancing Queen |-> <-| Mamamia |->


--------------------------------
Process exited after 23.56 seconds with return value 10
계속하려면 아무 키나 누르십시오 . . .
```

7.5 연결 리스트로 구현한 스택

- 연결리스트로 스택 구현
  - 스택의 크기를 동적으로 결정, 크기에 대한 제한이 없고 배열과 같이 스택 크기를 매우 크게 잡을 필요가 없다.
  - 반면, 동적 메모리 할당, 링크에 대한 처리로 인해 연산의 시간은 배열에 비해 더 걸릴 수 있다.



(a) 배열을 이용한 스택



(b) 연결 리스트를 이용한 스택

연결리스트 스택 정의

- 연결리스트 구현 스택의 ADT(외부 인터페이스)는 배열로 구현한 스택과 완전히 동일
  - 내부 연산 구현 등을 달라짐
  - 예) top은 정수가 아닌 포인터로 구현

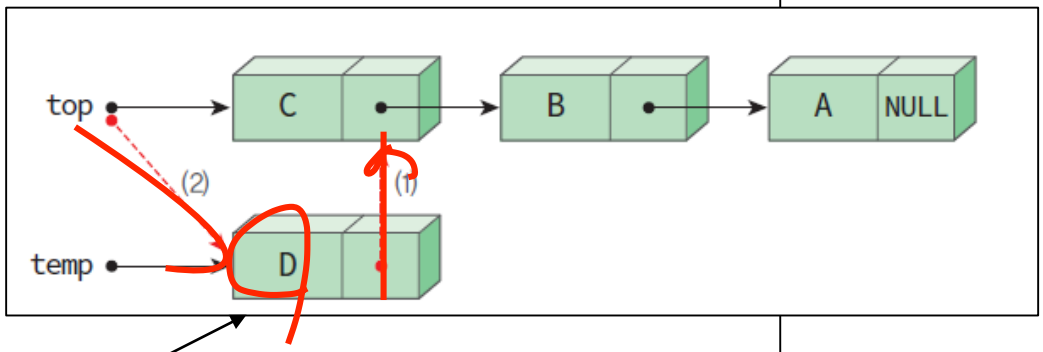
```
4   typedef int element;
5   typedef struct StackNode {
6       element data;
7       struct StackNode *link;
8   } StackNode;
9
10  typedef struct {
11      StackNode *top;
12  } LinkedStackType;
```

기존 배열 스택과 동일한 형식을 지원하기 위함

```
LinkedStackType s;
s->type = NULL;
```

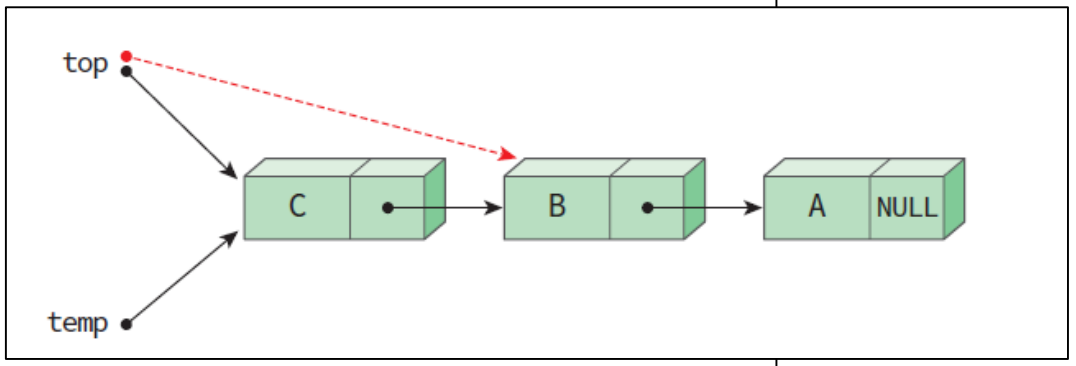
연결리스트 스택: linked_stack.c (1/3)


```
14  // 초기화 함수
15  void init(LinkedStackType *s)
16  {
17      s->top = NULL;
18  }
19
20  // 공백 상태 검출 함수
21  int is_empty(LinkedStackType *s)
22  {
23      return (s->top == NULL);
24  }
25
26  // 포화 상태 검출 함수
27  int is_full(LinkedStackType *s)
28  {
29      return 0;
30  }
31
32  // 삽입 함수
33  void push(LinkedStackType *s, element item)
34  {
35      StackNode *temp = (StackNode *)malloc(sizeof(StackNode));
36      temp->data = item;
37      temp->link = s->top;
38      s->top = temp;
39  }
```



연결리스트 스택: linked_stack.c (2/3)


```
41  // 삭제 함수
42  element pop(LinkedStackType *s)
43  {
44      element data;
45      if (is_empty(s)) {
46          fprintf(stderr, "스택이 비어있음\n");
47          exit(1);
48      }
49      else {
50          StackNode *temp = s->top;
51          data = temp->data;
52          s->top = s->top->link;
53          free(temp);
54          return data;
55      }
56  }
57
58  // 피크 함수
59  element peek(LinkedStackType *s)
60  {
61      if (is_empty(s)) {
62          fprintf(stderr, "스택이 비어있음\n");
63          exit(1);
64      }
65      else {
66          return s->top->data;
67      }
68  }
```



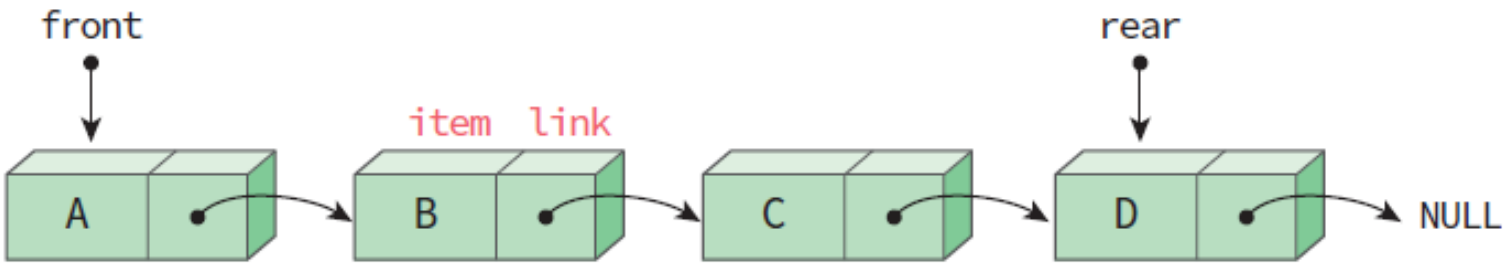
연결리스트 스택: linked_stack.c (3/3)


```c
void print_stack(LinkedStackType *s)
{
    StackNode *p;
    for (p = s->top; p != NULL; p = p->link)
        printf("%d->", p->data);
    printf("NULL \n");
}

// 주 테스트 함수
int main(void)
{
    LinkedStackType s;
    init(&s);
    push(&s, 1); print_stack(&s);
    push(&s, 2); print_stack(&s);
    push(&s, 3); print_stack(&s);
    pop(&s); print_stack(&s);
    pop(&s); print_stack(&s);
    pop(&s); print_stack(&s);
    return 0;
}
```

```
1->NULL
2->1->NULL
3->2->1->NULL
2->1->NULL
1->NULL
NULL

--------------------------------
Process exited after 0.0265 seconds with return value 0
계속하려면 아무 키나 누르십시오 . . .
```

7.5 연결 리스트로 구현한 큐

- 연결리스트로 구현한 스택과 동일하게 연결리스트로 구현한 큐도 가능
  - 장점: 크기 제한이 없고, 배열처럼 크게 공간을 잡을 필요가 없음
  - 단점: 링크 필드에 의한 공간 낭비와 구현이 조금 복잡해 짐

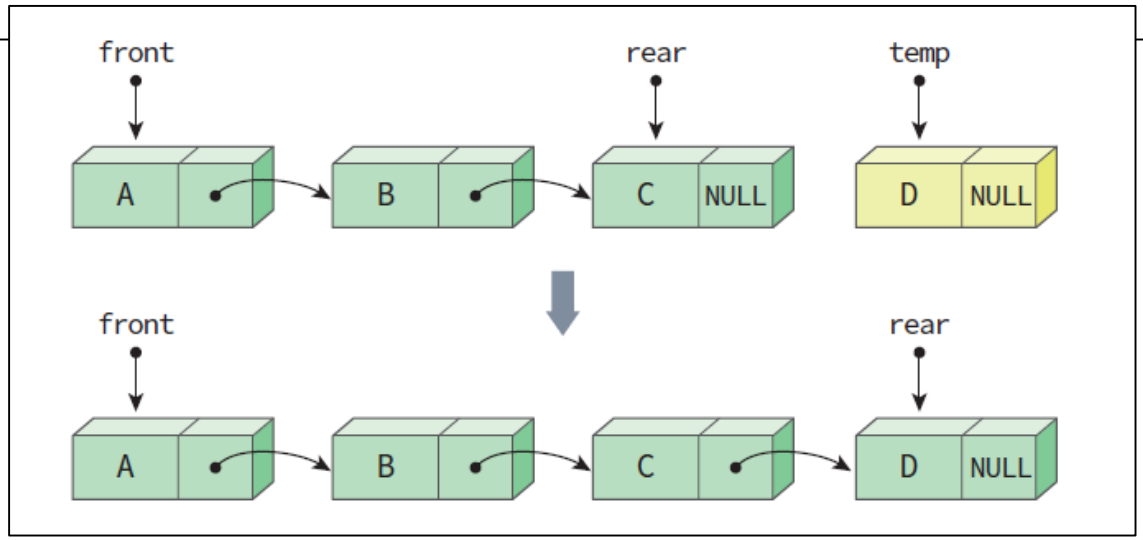


연결리스트 큐 구현: linked_queue.c (1/4)


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

typedef int element;          // 요소의 타입
typedef struct QueueNode {    // 큐의 노드의 타입
    element data;
    struct QueueNode *link;
} QueueNode;

typedef struct {              // 큐 ADT 구현
    QueueNode *front, *rear;
} LinkedQueueType;

// 큐 초기화 함수:
void init(LinkedQueueType *q)
{
    q->front = q->rear = 0;
}

// 공백 상태 검출 함수:
int is_empty(LinkedQueueType *q)
{
    return (q->front == NULL);
}

// 포화 상태 검출 함수
int is_full(LinkedQueueType *q)
{
    return 0;
}
```

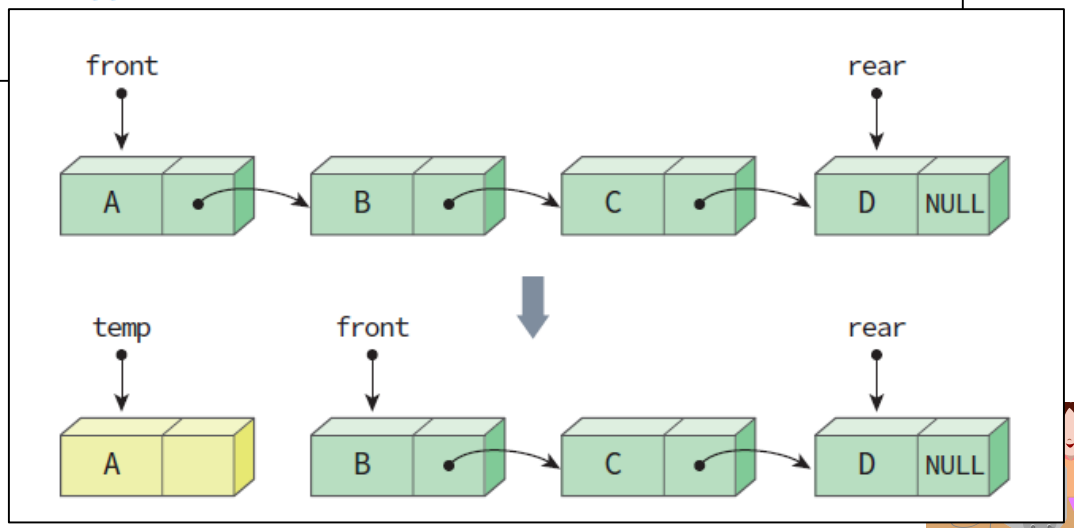
연결리스트 큐 구현: linked_queue.c (2/4)


```
33  void enqueue(LinkedQueueType *q, element data)
34  {
35      QueueNode *temp = (QueueNode *)malloc(sizeof(QueueNode));
36      temp->data = data;       // 데이터 저장
37      temp->link = NULL;       // 링크 필드를 NULL
38      if (is_empty(q)) {       // 큐가 공백이면
39          q->front = temp;
40          q->rear = temp;
41      }
42      else {       // 큐가 공백이 아니면
43          q->rear->link = temp;  // 순서가 중요
44          q->rear = temp;
45      }
46  }
```



연결리스트 큐 구현: linked_queue.c (3/4)


```
48  // 삭제 함수
49  element dequeue(LinkedQueueType *q)
50  {
51      QueueNode *temp =q-> front;
52      element data;
53      if (is_empty(q)) {       // 공백상태
54          fprintf(stderr, "스택이 비어있음\n");
55          exit(1);
56      }
57      else {
58          data = temp->data;          // 데이터를 꺼낸다.
59          q->front = q->front->link; // front를 다음노드를 가리키도록 한다.
60          if (q->front == NULL)       // 공백 상태
61              q->rear = NULL;
62          free(temp);                 // 동적메모리 해제.
63          return data;                // 데이터 반환
64      }
65  }
```



연결리스트 큐 구현: linked_queue.c (4/4)


```c
void print_queue(LinkedQueueType *q)
{
    QueueNode *p;
    for (p= q->front; p != NULL; p = p->link)
        printf("%d->", p->data);
    printf("NULL\n");
}

// 연결된 큐 테스트 함수
int main(void)
{
    LinkedQueueType queue;

    init(&queue);       // 큐 초기화

    enqueue(&queue, 1); print_queue(&queue);
    enqueue(&queue, 2); print_queue(&queue);
    enqueue(&queue, 3); print_queue(&queue);
    dequeue(&queue);        print_queue(&queue);
    dequeue(&queue);        print_queue(&queue);
    dequeue(&queue);        print_queue(&queue);
    return 0;
}
```

```
1->NULL
1->2->NULL
1->2->3->NULL
2->3->NULL
3->NULL
NULL

--------------------------------
Process exited after 0.638 seconds with return value 0
계속하려면 아무 키나 누르십시오 . . .
```