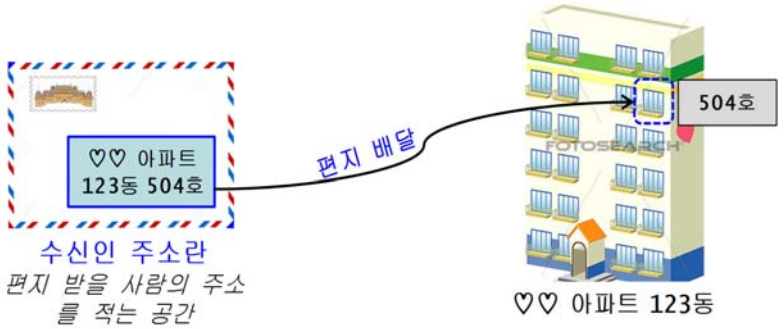
DATA
STRUCTURES
USING C

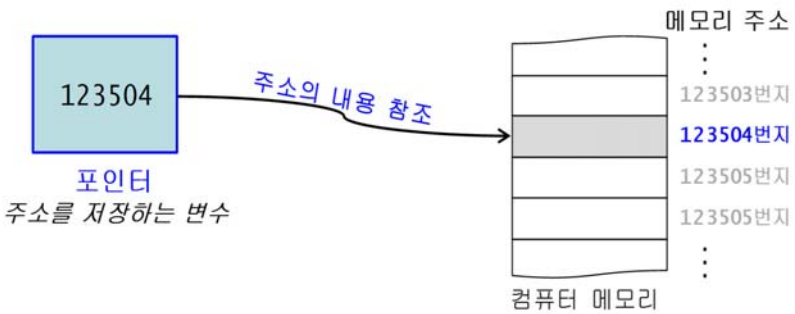
- 포인터의 개념을 이해한다.
- 포인터 관련 연산자를 이해하고 활용 능력을 기른다.
- 동적 메모리의 할당과 반납의 메커니즘을 이해한다.
- 연결 리스트의 개념을 이해한다.
- 스택과 큐를 연결 리스트로 구현하는 방법을 이해한다.
- 연결 리스트로 자료 구조를 구현하는 능력을 기른다.

CHAPTER 05

포인터와 연결리스트

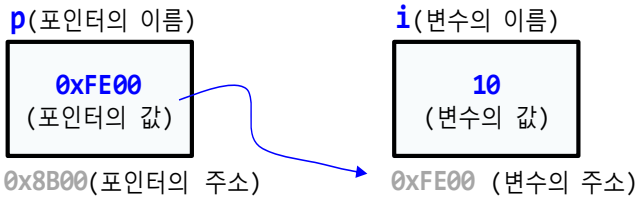
5.1 포인터(pointer)





포인터 선언

```
int i = 10;
int* p;
p = &i;
```



- 포인터 변수 선언

```
int* pi;        // int 변수를 가리킬 목적의 포인터 pi를 선언
float *pf;      // float 변수를 가리킬 목적의 포인터 pf를 선언
```

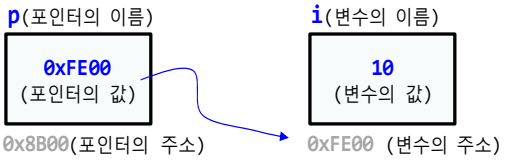
- 여러 개의 변수 선언

```
char* p, q, r;          // p는 char* 변수, q와 r은 char변수
char *p, *q, *r;        // p, q, r 모두 char*형 변수
```

- 사용하기 전에 반드시 초기화되어야 함

포인터 활용

```
int i = 10;
int* p;
p = &i;
```




```
*pi = 20;       // i=20과 동일
```

- *** 연산자** 사용의 여러가지 예

```
사용예#1:     c = a * b;
사용예#2:     int * p;
사용예#3:     *p = 20;
```

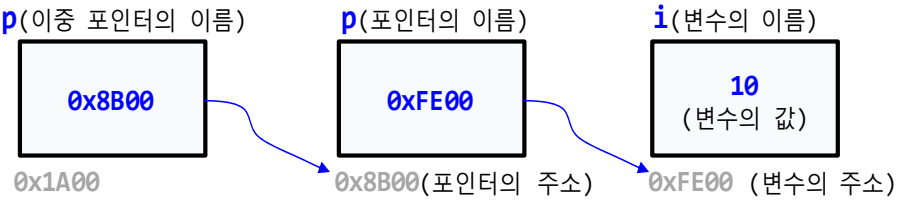
& 연산자
변수의 주소를 추출
*** 연산자**
포인터가 가리키는 곳의 내용을 추출

- 비트단위 AND **연산자 &**

```
사용예#1:     c = a & b;
사용예#2:     p = &i;
사용예#3:    int & p {i}; // C++에서만 사용 가능(참조자)
```

이중 포인터

```
int i = 10;
int* p = &i;
int** pp = &p;
```



표현	자료형	동일한 표현
10	int (상수)	
i	int	*p, **pp
p	int*	*pp, &i
pp	int**	&p

다양한 포인터

- 포인터의 종류

```
void *p;              // p는 아무것도 가리키지 않는 포인터
int *pi;              // pi는 정수 변수를 가리키는 포인터
float *pf;            // pf는 실수 변수를 가리키는 포인터
char *pc;             // pc는 문자 변수를 가리키는 포인터
int **pp;             // pp는 포인터를 가리키는 포인터
test *ps;             // test 타입의 객체를 가리키는 포인터
```

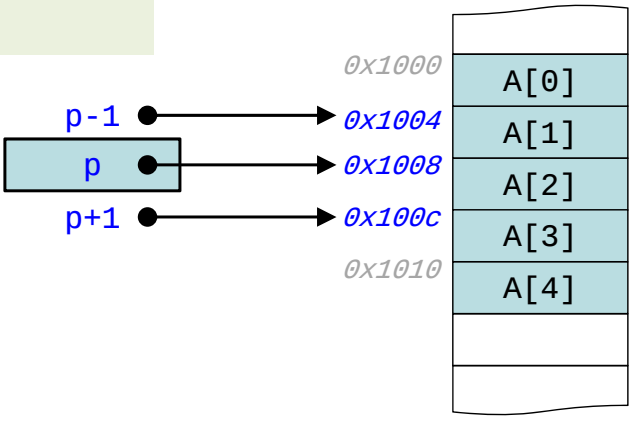
- `void (*f)(int) ;` // f는 함수를 가리키는 포인터

```
void *p;
pi=(int *) p;
```

포인터 연산

- 포인터에 대한 사칙연산
  - 포인터가 가리키는 객체단위로 계산된다.

```
int A[5], int *p;
p = &A[2];
p-1;
p+1;
```



포인터와 연산자

```
 p          // 포인터
*p          // 포인터가 가리키는 값
*p++        // 가리키는 값을 가져온 다음, 포인터를 한칸 증가
*p--        // 가리키는 값을 가져온 다음, 포인터를 한칸 감소
(*p)++      // 포인터가 가리키는 값을 증가시킨다.
```

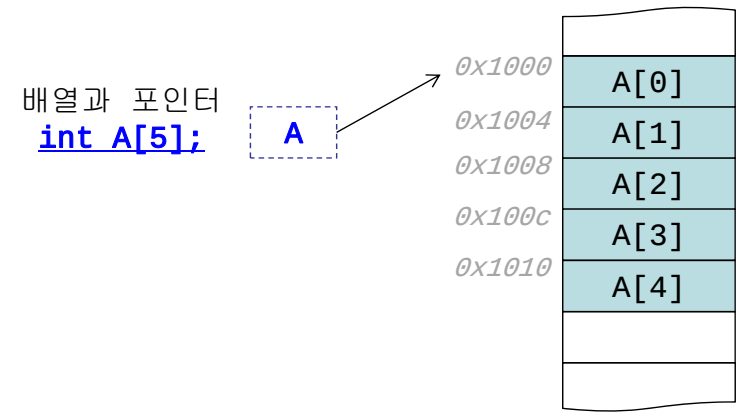
```
int a;      // 정수 변수 선언
int *p;     // 정수 포인터 선언
int **pp;   // 정수 포인터의 포인터 선언
p = &a;     // 변수 a와 포인터 p를 연결
pp = &p;    // 포인터 p와 포인터의 포인터 pp를 연결
```

문제

- 정수 변수의 주소를 출력하는 함수 print_address(int * px)를 작성하라. main()에서 정수 변수를 만들고 10으로 초기화한 후, 변수의 값과 주소를 출력한다. 다음으로 print_address()에 변수의 주소를 전달하고, 이 함수에서 주소와 그 주소에 저장된 정수 값을 다시 출력하도록 하라.

배열과 포인터

- 배열의 이름: 사실상의 포인터와 같은 역할
  - 컴파일러가 배열의 이름을 배열의 첫 번째 주소로 대치



문제

- 전달된 정수 배열의 모든 요소의 주소를 출력하는 함수 print_array(int a[i], int len)를 작성하라. 이때, len은 배열 a의 길이를 의미한다. main()에서 정수 배열을 만들고 배열에 적당한 값을 저장한 후에 print_array()에 해당 배열을 전달하라.

포인터와 구조체

```
typedef struct {
        int degree;
        float coef[MAX_DEGREE];
} Polynomial ;
```

```
Polynomial a;
Polynomial* p;
p = &a;
a.degree = 5;
p->coef[0] = 1;
```

- 구조체 멤버의 접근
  - 구조체에서는 "." 연산자
  - 포인터에서는 "->" 연산자

구조체를 이용한 표현		포인터를 이용한 표현	
a.degree	(&a)->degree	p->degree	(*p).degree
a.coef[0]	(&a)->coef[0]	p->coef[0]	(*p).coef[0]

포인터 기타

- **자기참조 구조체**

```
typedef struct ListNode {
        char data[10];
        struct ListNode* link;
} Node;
```

- **주의사항**
  - 포인터는 NULL로 초기화하는 것이 좋음
    - `int* pi=NULL;`
  - 초기화가 안 된 포인터 변수 → 접근하면 안됨.
    - char* pc; // 포인터 pc는 초기화가 안 되어 있음
    - *pc = 'a'; // 매우 위험한 코드
  - 포인터 사이의 변환에는 명시적인 형 변환
    - int * pi;
    - float * pf;
    - pf = (float*)pi;

문제

- 다음 프로그램의 출력은 무엇인가?

```
int value = 10, *temp;
temp = &value;
*temp = 20
printf("value = %d, *temp = %d₩n", value, *temp);
```

문제

- 다음의 함수 헤더를 사용하여 함수 swap()을 작성하라. swap() 함수를 사용하여 두 정수의 값을 서로 바꾸어보자.

```
void swap (int * p1, int * p2);
void main() {
    int x=10;
    int y=20;
    swap( ... );
    printf("x=%d, y=%d ₩n", x, y);
}
```


5.2 정적 메모리

- 정적 메모리 할당 (static memory allocation)
  - 메모리의 크기는 프로그램이 시작하기 전에 결정
  - 실행 도중에 크기를 변경할 수 없다.
  - 더 큰 입력이 들어온다면 처리하지 못함
  - 더 작은 입력이 들어온다면 메모리 공간 낭비

```
int i;              // int형 변수 i를 정적으로 할당
int* p;             // 포인터 변수 p를 정적으로 할당
int A[10];          // 길이가 10인 배열을 정적으로 할당
```

- 이것이 가능한가?

```
int n=10;
int B[n];          // 컴파일 오류.
```

동적 메모리 라이브러리 함수

- 실행 도중에 메모리를 할당 받는 것
  - 필요한 만큼만 할당을 받고 반납함
  - 메모리를 매우 효율적으로 사용가능

```
void* malloc(int size);
void* calloc(int num, int size);
```

```
void free(void* ptr)
```

- 포인터와 동적 메모리 할당
  - 동적으로 할당된 메모리는 반드시 포인터에 저장
  - 그래야 사용할 수도 있고 해제할 수도 있다.

동적 메모리 할당과 해제

```
int* pi = (int*)malloc(sizeof(int)*10);
Polynomial* pa = (Polynomial*)malloc(sizeof(Polynomial));
pi[3] = 10;      // 동적 할당된 메모리를 배열처럼 사용
pa->degree = 5; // 동적 할당된 다항식 구조체의 멤버 변경
...
free(pi);
free(pa);
```

문제

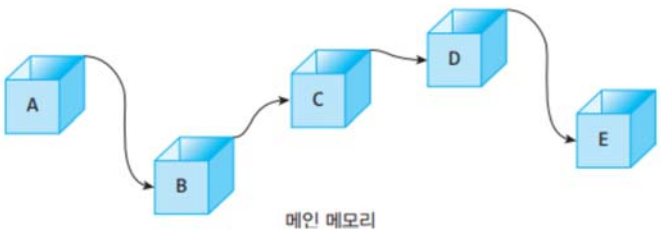
- 동적 메모리 할당을 이용하여 정수를 저장할 수 있는 동적 배열을 생성하여 다음과 같은 메뉴를 실행하는 프로그램을 작성하라.

  ==========
  1. 크기가 n인 동적 배열을 생성
  2. 배열을 난수로 채운다.
  3. 배열의 각 원소를 출력한다.
  4. 배열의 각 원소들의 합을 출력한다.
  5. 동적 할당 공간을 반납한다.

  ==========
  메뉴를 선택하시오:

5.3 포인터의 응용 : 연결된 표현

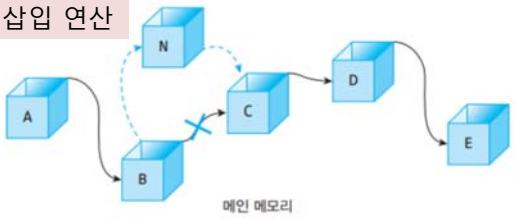
- **Linked Representation ←→ 배열**
  - 항목들을 노드(node)라고 하는 곳에 분산하여 저장
  - 다음 항목을 가리키는 주소도 같이 저장
    - 노드 (node) : <항목, 주소> 쌍
  - 노드는 데이터 필드와 링크 필드로 구성
    - 데이터 필드 – 리스트의 원소, 즉 데이터 값을 저장하는 곳
    - 링크 필드 – 다른 노드의 주소값을 저장하는 장소 (포인터)
  - 메모리안에서의 노드의 물리적 순서가 리스트의 논리적 순서와 일치할 필요 없음



연결된 표현의 장단점

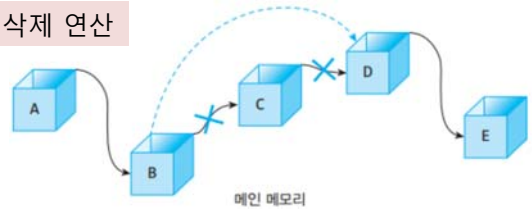
- 장점
  - 삽입, 삭제가 보다 용이하다.
  - 연속된 메모리 공간이 필요없다.
  - 크기 제한이 없다

삽입 연산



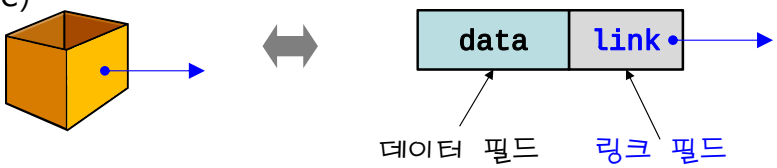
- 단점
  - 구현이 어렵다.
  - 오류가 발생하기 쉽다.

삭제 연산

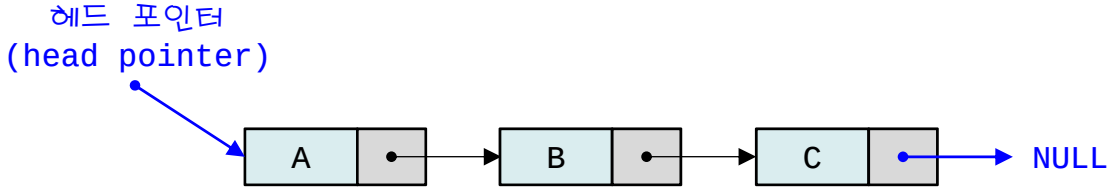


연결 리스트의 구조

- 노드(node)

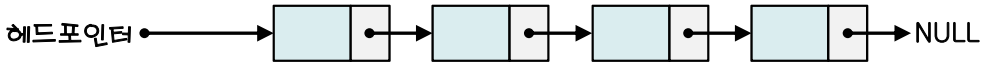


- 헤드 포인터(head pointer)
  - 리스트의 첫 번째 노드를 가리키는 변수

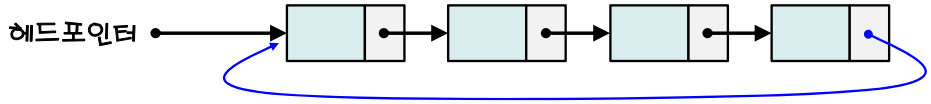


연결 리스트의 종류

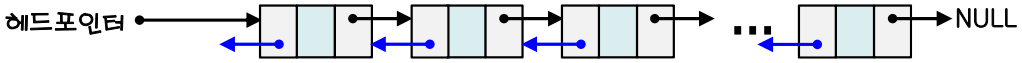
- 단순 연결 리스트(singly linked list)



- 원형 연결 리스트(circular linked list)



- 이중 연결 리스트(doubly linked list)



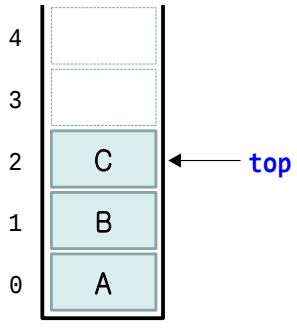
5.4 연결 리스트로 구현한 스택

- 노드 구조체

```
typedef int Element;
typedef struct LinkedNode {
        Element data;
        struct LinkedNode* link;
} Node;
```

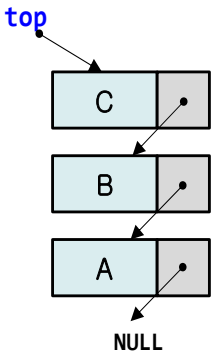
- 배열과 연결리스트를 이용한 스택의 비교

```
Element data[MAX_STACK_SIZE]
int top;
```



배열을 이용한 스택

```
Node* top;
```

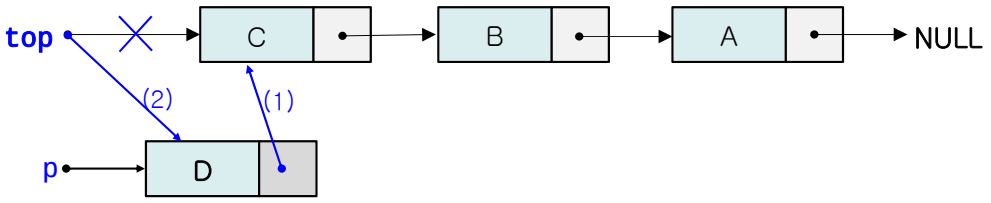


연결 리스트를 이용한 스택

삽입 연산

① 노드 D의 링크 필드가 노드 C를 가리키도록 한다: p->link = top;

② 헤더 포인터 top이 노드 D를 가리키도록 한다: top = p;




```
void push( Node *p )
{
  p->link = top; //(1)
  top = p;       //(2)
}
```

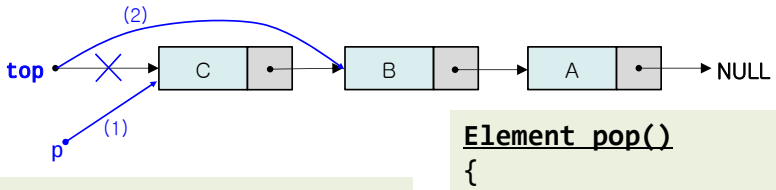
```
void push(Element e)
{
  Node* p = (Node*)malloc(sizeof(Node));
  p->data = e;
  p->link = top; // (1)
  top = p;       // (2)
}
```


삭제 연산

① 포인터 변수 p가 노드 C를 가리키도록 한다: p = top;

② 헤더 포인터 top이 B를 가리키도록 한다: top = p->next;

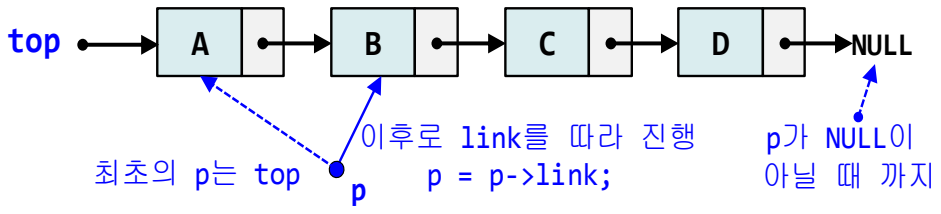
③ 마지막으로 변수 p의 값을 반환한다: return p;




```
Node* pop()
{
   Node* p;
   if(is_empty()) error("에러");

   p = top;
   top = p->link;

   return p;
}
```

```
Element pop()
{
  Node* p;
  Element e;
  if (is_empty()) error("에러");

  p = top;       // (1)
  top = p->link; // (2)

  e = p->data;
  free(p);
  return e;
}
```

전체 노드 방문 연산




```
int size()
{
        Node *p;
        int count = 0;

        for (p = top; p != NULL; p = p->link)
                count++;

        return count;
}
```


프로그램 5.5 연결된 스택 테스트 프로그램

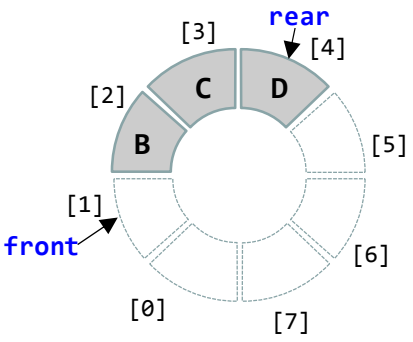
```
typedef int Element;
typedef struct LinkedNode {
Element data;// 데이터 영역
struct LinkedNode* link;// 다음 노드를 가리키는 포인터 변수
} Node;
Node* top = NULL;// 연결 리스트의 헤드 포인터
```

- 연결 리스트로 스택을 구현하는 경우의 장점은?
- 연결 리스트로 구현된 스택의 경우 top 변수의 의미는 무엇인가?
- 다음 중 연결 리스트로 구현된 스택에서 공백 상태에 해당하는 조건은?
  1. top == NULL
  2. *top == NULL
  3. *top == MAX_STACK_SIZE
  4. *top == MAX_STACK_SIZE-1

5.5 연결리스트로 구현한 큐

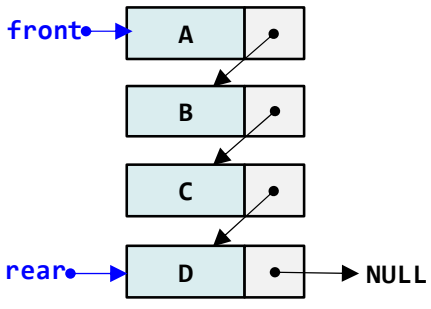
- 배열과 연결리스트를 이용한 큐의 비교

```
Element data[MAX_QUEUE_SIZE]
int front;
int rear;
```



배열을 이용한 원형 큐

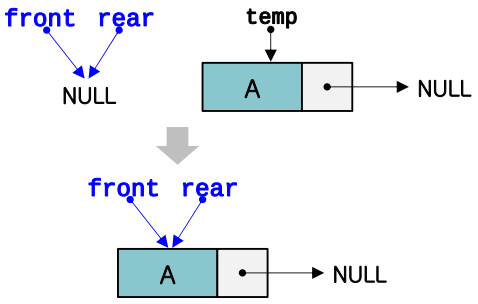
```
Node* front;
Node* rear;
```



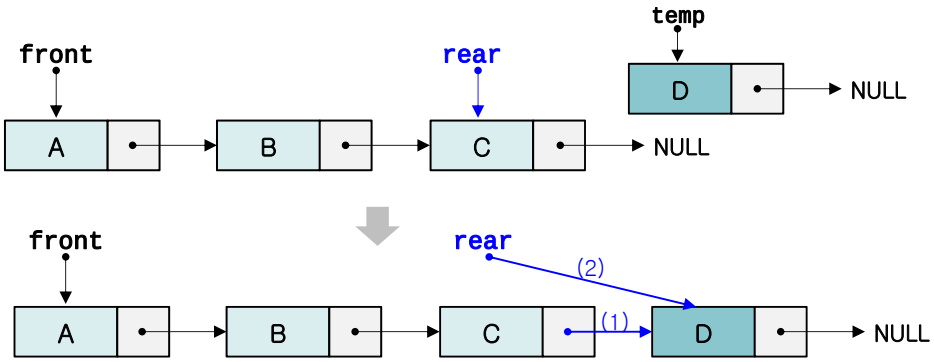
연결 리스트를 이용한 큐

삽입 연산

- 공백상태의 삽입



- 비 공백상태의 삽입

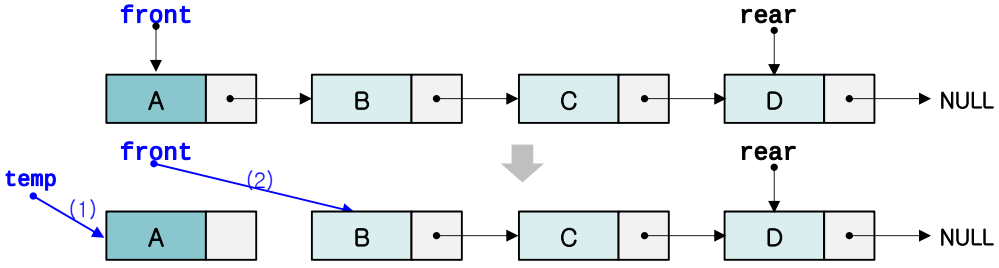


프로그램 5.6 연결된 큐의 삽입 연산

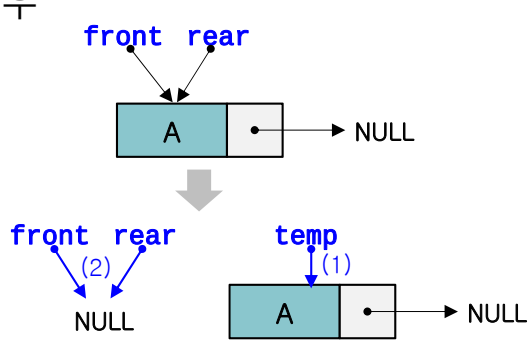
```
void enqueue(Element e)
{
Node* p = (Node*)malloc(sizeof(Node));// 노드 동적 할당
p->data = e;                          // 데이터 필드 초기화
p->link = NULL;                       // 링크 필드 초기화

if (is_empty()) front = rear = p;// 그림 5.16의 (a)
else {
rear->link = p;// 그림 5.16 (b)의 (1)
rear = p;// 그림 5.16 (b)의 (2)
}
}
```

삭제 연산

- 노드가 두 개 이상



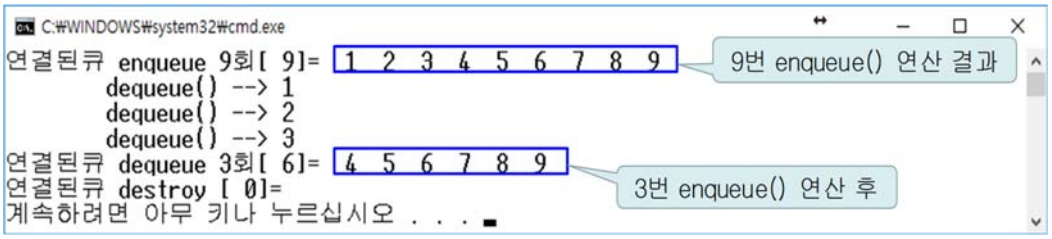
- 노드가 하나인 경우



연결된 큐의 구현

```
typedef int Element;
typedef struct LinkedNode {
        Element data;
        struct LinkedNode* link;
} Node;
Node* front = NULL;
Node* rear = NULL;
```

- 실행 결과

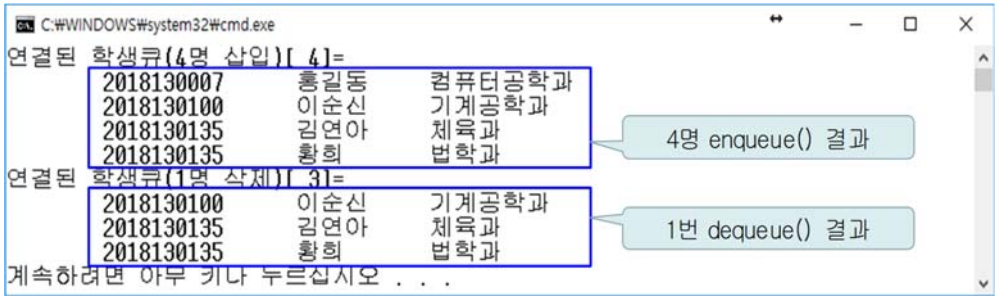


학생 정보 큐

- 구조체 선언 및 Element 정의

```
typedef struct Student_t {
        int     id;
        char    name[32];
        char    dept[32];
} Student;
// Student 대신에 Element를 사용할 수 있음
typedef Student Element;
```

- 실행결과



문제

- 사용자로부터 양의 정수를 입력받아서 연결된 큐에 저장하고, 결과를 다음과 같이 출력하는 프로그램을 작성하라.

  양의 정수를 입력하세요(종료:-1): 10
  양의 정수를 입력하세요(종료:-1): 20
  양의 정수를 입력하세요(종료:-1): 30
  양의 정수를 입력하세요(종료:-1): -1
  10->20->30->NULL