[4장 연습문제]

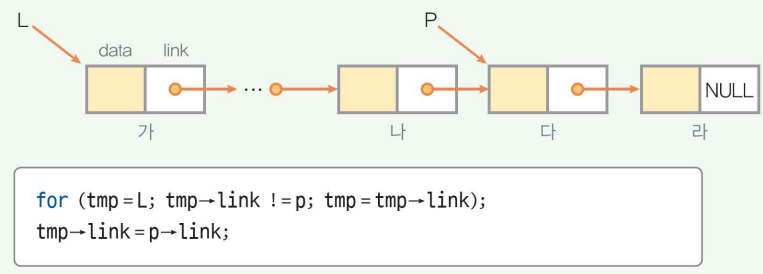
01. 연결 리스트를 사용하기에 적합한 경우는?

① 자료를 정렬하는 경우
② 자료를 역순으로 처리하는 경우
③ 자료의 삽입과 삭제가 많은 경우
④ 자료를 탐색하는 경우

02. 연결 리스트에 대한 설명으로 거리가 먼 것은?

① 노드의 삽입과 삭제가 쉽다.
② 노드들이 포인터로 연결되어 있어 탐색이 빠르다.
③ 연결해 주는 포인터를 위한 추가 공간이 필요하다.
④ 연결 리스트 중에서 중간 노드 연결이 끊어지면 그 다음 노드를 찾기 어렵다.

03. 다음과 같은 단순 연결 리스트에 대해, 아래와 같은 C 언어로 작성된 프로그램을 수행한 후 포인터 tmp가 가리키는 노드는?



① 가
② 나
③ 다
④ 라

04. n개의 데이터로 구성된 선형 리스트를 단순 연결 리스트로 표현하고자 한다. 다음 중 시간 복잡도가 가장 낮은 연산은?

① 포인터가 가리키는 노드의 다음 노드를 리스트에서 삭제
② 리스트 길이를 출력
③ 포인터값이 주어진 임의의 노드 앞에 새로운 노드 추가
④ 마지막 노드의 데이터

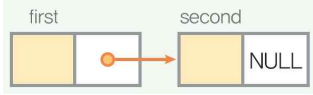
05. 자료들이 단순 연결 리스트에 다음과 같이 구성되어 있을 때, 자료 B를 삭제한 후 변경된 내용으로 옳은 것은?

메모리 주소	data	link
2000	A	2020
2010	B	2030
2020	C	2010
2030	D	0

① A의 link → 2030
② B의 link → 2010
③ B의 link → 2020
④ C의 link → 2030

06. 다음의 왼쪽은 연결 리스트를 만들기 위한 코드의 일부분이다. 오른쪽 그림과 같이 두 노드 first와 second가 연결되었다고 가정하고 왼쪽 코드를 참조하여 노드 tmp를 노드 first와 노드 second 사이에 삽입할 때의 코드로 옳은 것은?

```
struct node {
    int number;
    struct node *link;
};
struct node first;
struct node second;
struct node tmp;
```



① tmp.link = &first;
first.link = &tmp;

② tmp.link = first.link;
first.link = &tmp;

③ tmp.link = &second;
first.link = second.link;

④ tmp.link = NUL;
second.link = &tmp;

07. 다음 C 코드에서 foobar 함수가 하는 일을 바르게 설명한 것은?

```c
struct list_t {
   int data;
   struct list_t *link;
};

typedef struct list_t *list_p;

void foobar(list_p *ptr, struct list_t *node) {
   if (*ptr == NULL) {
      *ptr = node;
      node->link = node;
   }
   else {
      node->link = (*ptr)->link;
      (*ptr)->link = node;
   }
}
```

① 한 줄로 연결된 단순 연결 리스트에서 *ptr이 가리키는 노드의 뒤에 node를 삽입, 리스트가 비어 있으면 node로 새로 리스트를 만든다.
② 한 줄로 연결된 이중 연결 리스트에서 *ptr이 가리키는 노드의 뒤에 node를 삽입, 리스트가 비어 있으면 node로 새로 리스트를 만든다.
③ 원형으로 연결된 단순 연결 리스트에서 *ptr이 가리키는 노드의 뒤에 node를 삽입, 리스트가 비어 있으면 node로 새로 리스트를 만든다.
④ 원형으로 연결된 이중 연결 리스트에서 *ptr이 가리키는 노드의 뒤에 node를 삽입, 리스트가 비어 있으면 node로 새로 리스트를 만든다.

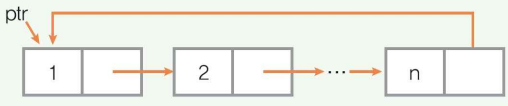
08. 다음 알고리즘은 주어진 단순 연결 리스트를 역순으로 변환하는 알고리즘이다. 알고리즘의 ㉠에 들어갈 내용으로 옳은 것은? (단, 리스트의 시작 주소를 나타내는 포인터는 start이며 노드의 연결 포인터 필드는 link이다.)

```
p = start;
q = NULL;
while (p != NULL) {
   ㉠
   p = p -> link;
   q -> link = r;
}
start = q;
```

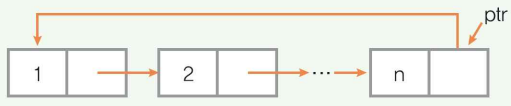
① q = q -> link; r = q;
② r = q; q = p;
③ r = q; q = p -> link;
④ q = q -> link;

09. 원형 연결 리스트 맨 앞에 새로운 노드를 삽입할 때, ㉠, ㉡에 대한 시간 복잡도는?

㉠ 포인터 ptr이 원형 연결 리스트의 맨 앞 노드를 가리키는 경우

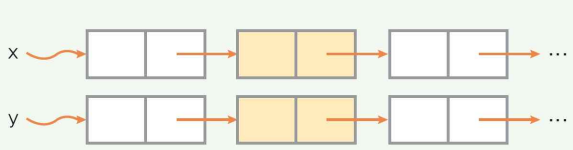


㉡ 포인터 ptr이 원형 연결 리스트의 맨 뒤 노드를 가리키는 경우



	㉠	㉡
①	O(1)	O(1)
②	O(1)	O(n)
③	O(n)	O(1)
④	O(n)	O(n)

10 구조체 list를 이용한 단순 연결 리스트 x, y의 후위 노드(색칠된 부분)들을 서로 스왑하기 위한 코드는? (단, 후위 노드들의 next는 NULL이 아니다.)




```
struct list {
   int data;
   struct list *next;
} *x, *y, *tmp;
```

①
```
tmp = x->next;
x->next->next = y->next->next;
y->next->next = tmp;
tmp = y->next;
x->next = y->next;
y->next = tmp;
```

②
```
tmp = x->next->next;
x->next->next = y->next->next;
y->next->next = tmp;
tmp = x->next;
x->next = y->next;
y->next = tmp;
```

③
```
tmp = x->next;
x->next = y->next;
y->next = tmp;
tmp = x->next;
x->next->next = y->next->next;
y->next->next = tmp;
```

④
```
tmp = x->next->next;
y->next->next = tmp;
x->next->next = y->next->next;
tmp = x->next;
y->next = tmp;
x->next = y->next;
```

11. 희소행렬을 연결 리스트로 표현할 때 가장 큰 장점은?

① 기억 장소가 절약된다.
② 임의의 위치로 액세스가 가능하다.
③ 이진 탐색이 가능하다.
④ 행렬 사이의 연산 시간을 줄일 수 있다.

12. 이중 연결 리스트에서 노드 p 다음(next) 노드가 q이고 노드 q 이전(prev) 노드가 p인 경우, 인접한 p, q 노드의 위치를 서로 바꾸는 연산 순서로 옳은 것은? (단, p, q 노드는 모두 연결 리스트의 처음 노드나 마지막 노드가 아니라고 가정한다.)

```
ㄱ. p → prev → next = q;
ㄴ. q → next → prev = p;
ㄷ. p → prev = q;
ㄹ. q → next = p;
ㅁ. q → prev = p → prev;
ㅂ. p → next = q → next;
```

① ㄱ - ㄴ - ㄷ - ㄹ - ㅁ - ㅂ
② ㄱ - ㄴ - ㅁ - ㅂ - ㄷ - ㄹ
③ ㄱ - ㄴ - ㅂ - ㄹ - ㄷ - ㅁ
④ ㄷ - ㄹ - ㅁ - ㅂ - ㄱ - ㄴ

13. 원형 연결 리스트 길이를 계산하기 위한 C 함수를 작성하려고 한다. ㉠과 ㉡에 들어갈 알맞은 명령은? (단, 원형 연결 리스트의 길이는 노드 개수로 정의하고, 원형 연결 리스트의 시작 포인터는 ptr이라고 가정한다.)

```
struct list_node {
   char data;
   struct list_node * link;
}

typedef struct list_node * list_pointer;

int CListLength(list_pointer ptr) {
   list_pointer temp;
   int count = 0;
   if ( ㉠ ) {
      temp = ptr;
      do {
         count++;
      } while ( ㉡ );
   }
   return count;
}
```

	㉠	㉡
①	!ptr	temp -> link != ptr
②	ptr	temp=temp -> link, temp != ptr
③	!ptr	temp=temp -> link, temp == ptr
④	ptr	temp -> link == ptr

14. 이중 연결 리스트에서 포인터 p가 가리키는 노드의 오른쪽에 포인터 newNode가 가리키는 노드를 삽입할 때, 연산 순서가 바르게 나열된 것은? (단, llink는 왼쪽 노드를 가리키는 포인터이고 rlink는 오른쪽 노드를 가리키는 포인터이다.)

ㄱ. p –〉 rlink = newNode;
ㄴ. newNode –〉 llink = p;
ㄷ. newNode –〉 rlink = p –〉 rlink;
ㄹ. p –〉 rlink –〉 llink = newNode;

① ㄱ - ㄴ - ㄷ - ㄹ
② ㄴ - ㄱ - ㄹ - ㄷ
③ ㄴ - ㄷ - ㄱ - ㄹ
④ ㄴ - ㄷ - ㄹ - ㄱ

15. 순차 자료구조와 연결 자료구조를 비교하여 설명하시오.

16. 단순 연결 리스트, 원형 연결 리스트, 이중 연결 리스트의 특징을 비교하여 설명하시오.

17. 다음은 원형 연결 리스트의 맨 앞에 새로운 노드를 추가하는 알고리즘의 가상코드이다. ㉠, ㉡에 들어갈 문장으로 옳은 것은? (단, 원형 연결 리스트에서 last는 리스트의 맨 마지막 노드를 가리키는 포인터이고 초깃값은 NULL 가상코드이며, 변수 p는 새로 삽입되는 노드를 나타낸다. 또한 리스트에 노드를 세 개 이상 추가할 수 있어야 한다.)

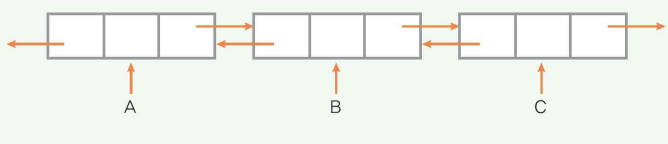
```
struct node {
   int key;
   struct node *next;
} *last = NULL;                // 마지막 노드를 가리키는 포인터

Function insertFront(struct node *p)
   if (last == NULL) then      // 리스트가 빈 경우
      last = p;
      p→next = p;
   else                        // 리스트에 하나 이상의 노드가 있는 경우
      ㉠
      ㉡
   end if
return;
end
```

	㉠	㉡
①	p → next = last → next;	last → next = p;
②	last → next = p;	p → next = last → next;
③	p → next = last;	last → next = p;
④	last → next = p;	p → next = last;

18. 다음 구조체를 갖는 이중 연결 리스트에서 A, B, C는 각각 노드를 가리키는 포인터 변수이다. 노드 B를 삭제하기 위한 명령으로 옳지 않은 것은?

```
struct Dlist {
   int data;
   struct Dlist *left;
   struct Dlist *right;
};
```



① B -> right -> left = B -> left;
B -> left -> right = B -> right;

② A -> right = C;
C -> left = A;

③ A -> right = A -> right -> right;
B -> right -> left = B -> left;

④ C -> left -> right = B -> right;
C -> left = B -> left;

19. 다음은 이중 연결 리스트의 맨 앞에 새로운 키값을 갖는 노드를 삽입하는 알고리즘이다. ㉠~㉢에 들어갈 문장으로 바르게 짝지어진 것은? (단, head는 이중 연결 리스트의 맨 앞 노드를 가리키는 포인터이며, 리스트는 초기에 비어 있다고 가정한다.)

```
struct node {                    // 이중 연결 리스트의 노드 구조
   int key;
   struct node *prev;            // 이전 노드를 가리키는 포인터
   struct node *next;            // 다음 노드를 가리키는 포인터
};

struct node *head = NULL;        // 맨 처음 노드를 가리키는 포인터

void insertDoubleList(int ikey) {
   struct node *temp;
   temp = (struct node *) malloc(sizeof(struct node));
   if (temp == NULL) {
      printf("Memory allocation is failed. \n");
      exit(1);
   }
   temp→key = ikey;
   ㉠ ;
   if (head == NULL)             // 리스트에 노드가 없는 경우
      ㉡ ;
   else {                        // 리스트에 노드가 있는 경우
      ㉢ ;
      head→prev = temp;
   }
   head = temp;
}
```

	㉠	㉡	㉢
①	temp → prev = NULL	temp → prev = head	temp → prev = head
②	temp → prev = NULL	temp → next = NULL	temp → next = head
③	head → next = temp	temp → next = NULL	temp → prev = head
④	head → next = temp	temp → prev = head	temp → next = head

20. 다음과 같이 구조체 자료형인 _node를 선언하고 이를 이용해 연결 리스트를 만들었다. 다음 코드를 보고 물음에 답하시오(단 시작 함수는 _tmain( )이다).

(가) 숫자 10, 5, 8, 3, 1, 7을 삽입하되 작은 수부터 연결 리스트가 유지되도록 함수 ordered_insert(int k)를 작성하시오(단, k는 삽입하려는 정수이다).

(나) 연결 리스트를 구성하는 각 node의 변수 data를 모두 출력하는 함수 print_list(node* t)를 작성하시오(단, t는 node에 대한 시작 포인터이고, 화면에 출력할 함수는 printf()를 사용한다).

(다) 삭제하려는 숫자를 인수로 받아 그 노드를 삭제하는 함수 delete_node(int k)를 작성하시오(단, k는 삭제하려는 정수이다).

```
typedef struct _node {
   int data;
   struct _node *next;
} node;

node *head, *tail;

void init_list(void) {
   head = (node*)malloc(sizeof(node));
   tail = (node*)malloc(sizeof(node));
   head->next = tail;
   tail->next = tail;
}

node *ordered_insert(int k) {}
node *print_list(node* t) {}
int delete_node(int k) {}

int _tmain(int argc, _TCHAR* argv[]) {
   node *t;

   init_list();
   ordered_insert(10);
   ordered_insert(5);
   ordered_insert(8);
   ordered_insert(3);
   ordered_insert(1);
   ordered_insert(7);

   printf("\nInitial Linked list is ");
   print_list(head->next);

   delete_node(8);
   print_list(head->next);

   return 0;
}
```