# 01. 다음 중 NULL 포인터(NULL pointer)가 존재하지 않는 구조는 어느 것인가?

(1) 단순 연결 리스트 (2) 원형 연결 리스트  
 (3) 이중 연결 리스트 (4) 헤더 노드를 가지는 단순 연결 리스트

NULL 포인터는 연결 리스트 노드의 끝을 나타내기 위해 있는데, 원형 연결 리스트는 마지막 노드가 첫 번째 노드를 가리키고 있기 때문에 NULL 포인터가 존재하지 않는다.

# 02. 리스트의 *n*번째 요소를 가장 빠르게 찾을 수 있는 구현 방법은 무엇인가?

(1) 배열 (2) 단순 연결 리스트  
 (3) 원형 연결 리스트 (4) 이중 연결 리스트

배열에서 n번째 요소를 찾을 때 시간 복잡도가 O(1)인 것에 반해 연결 리스트는 시간 복잡도가 O(n)이므로 배열이 n번째 요소를 가장 빠르게 찾을 수 있다.

# 03. 단순 연결 리스트에서 포인터 last가 마지막 노드를 가리킨다고 할 때 다음 수식 중, 참인 것은?

(1) last == NULL (2) last->data == NULL  
 (3) last->link == NULL (4) last->link->link == NULL

단순 연결 리스트에서 마지막 노드의 포인터는 NULL을 가리키고 있어야 하기에 마지막 노드의 link 부분에는 NULL이 저장되어 있어야 한다.

# 04. 단순 연결 리스트의 노드들을 포인터 p로 방문하고자 한다. 현재 p가 가리키는 노드에서 다음 노드로 가려면 어떤 코드를 사용해야 하는가?

(a) p++; (b) p--;  
 (c) p=p->link; (d) p=p->data;

노드에서 link 부분이 다음 노드의 주소를 가지고 있다.

# 05. 다음과 같이 변수 p가 2를 저장하는 노드를 가리키도록 하는 문장을 작성하라.

p=p->link;

# 06. 다음과 같이 변수 q가 1를 저장하는 노드를 가리키도록 하는 문장을 작성하라.

q=p;

# 07. 다음과 같은 연결 리스트에 아래와 같은 코드를 실행한다고 하자. 실행이 끝난 후에 포인터 p가 가리키는 노드는 어떤 노드인가?

D

# 08. 덱(deque: double-ended queue)은 삽입과 삭제가 양끝에서 임의로 수행되는 자료 구조이다. 다음 그림과 같이 단순 연결 리스트(singly linked list)로 구현한다고 할 때 O(1) 시간 내에 수행할 수 없는 연산은? (단, first와 last는 각각 덱의 첫 번째 원소와 마지막 원소를 가리키며, 연산이 수행된 후에도 덱의 원형이 유지되어야 한다)(국가시험 기출문제)

① insertFirst 연산: 덱의 첫 번째 원소로 삽입  
② insertLast 연산: 덱의 마지막 원소로 삽입  
③ deleteFirst 연산: 덱의 첫 번째 원소를 삭제  
④ deleteLast 연산: 덱의 마지막 원소를 삭제

단순 연결 리스트가 front와 last 포인터들에 대한 정보를 가지고 있다는 가정 하에, insertFirst, insertLast, 그리고 deleteFirst는 포인터들이 가리키고 있는 노드에서 바로 수행이 될 수 있지만, deleteLast는 삭제되는 노드 앞에 있는 노드를 가리키고 있는 포인터를 활용해서 수행되기 때문에 O(1) 시간 복잡도를 가질 수 없다.

# 09. 다음과 같이 단순 연결 리스트에 사용자가 입력하는 값을 저장했다가 출력하는 프로그램을 작성하라.

『===================(단순 연결 리스트 알고리즘) ===================』  
int main(void) {  
 ListNode \*head = NULL;  
 int len, data;  
  
 printf(“노드의 개수 : “);  
 scanf\_s(“%d”, &len);  
  
 int arr\_data[len];  
 for (int i = 0; i < len; i++) {  
 printf(“노드 #%d 데이터 : “, i + 1);  
 scanf\_s(“%d”, &data);  
 arr\_data[i] = data;  
 }  
 for (int i = len; i > 0; i--) {  
 insert\_first(head, arr\_data[i - 1]);  
 }  
  
 printf(“생성된 연결 리스트 : “);  
 print\_list(head);  
}

# 10. 다음과 같이 단순 연결 리스트의 노드들의 개수를 계산하는 프로그램을 작성해보자.

『===================(단순 연결 리스트 알고리즘) ===================』  
int main(void) {  
 ListNode \*head = NULL;  
 int len, data;  
  
 printf(“노드의 개수 : “);  
 scanf\_s(“%d”, &len);  
  
 int arr\_data[len];  
 for (int i = 0; i < len; i++) {  
 printf(“노드 #%d 데이터 : “, i + 1);  
 scanf\_s(“%d”, &data);  
 arr\_data[i] = data;  
 }  
 for (int i = len; i > 0; i--) {  
 insert\_first(head, arr\_data[i - 1]);  
 }  
  
 int count = 0;  
 for (ListNode \*p = head; p != NULL; p = p->link) {  
 count++;  
 }  
 printf(“연결 리스트 노드의 개수 = %d“, count);  
}

# 11. 단순 연결 리스트에 정수가 저장되어 있다. 연결 리스트에 있는 모든 노드의 데이터 값을 합한 결과를 출력하는 프로그램을 작성하시오.

『===================(단순 연결 리스트 알고리즘) ===================』  
int main(void) {  
 ListNode \*head = NULL;  
 int len, data;  
  
 printf(“노드의 개수 : “);  
 scanf\_s(“%d”, &len);  
  
 int arr\_data[len];  
 for (int i = 0; i < len; i++) {  
 printf(“노드 #%d 데이터 : “, i + 1);  
 scanf\_s(“%d”, &data);  
 arr\_data[i] = data;  
 }  
 for (int i = len; i > 0; i--) {  
 insert\_first(head, arr\_data[i - 1]);  
 }  
  
 int sum = 0;  
 for (ListNode \*p = head; p != NULL; p = p->link) {  
 sum += p->data;  
 }  
 printf(“연결 리스트의 데이터 합 : %d“, sum);  
}

# 12. 연결 리스트에서 특정한 데이터 값을 갖는 노드의 개수를 계산하는 함수를 작성하라.

element find\_data(ListNode \*head, element find) {  
 int count = 0;  
 ListNode \*p = head;  
 while (p != NULL) {  
 if (p->data == find) count++;  
 p = p->link;  
 }  
}

# 13. 단순 연결 리스트에서의 탐색함수를 참고하여 특정한 데이터 값을 갖는 노드를 삭제하는 함수를 작성하라.

ListNode\* delete\_specific(ListNode \*head, element find) {  
 ListNode \*p = NULL;  
 ListNode \*removed = head;  
 while (removed->data != find) {  
 p = removed;  
 removed = removed->link;  
 p = p->link;  
 }  
 if (p == NULL) {  
 head = head->link;  
 } else {  
 p->link = removed->link;  
 }  
 free(removed);  
 return head;  
}

# 14. 다음 그림과 같은 데이터를 저장할 수 있는 단순 연결 리스트를 생성하는 프로그램을 작성해보자.

typedef struct ListNode {  
 char name[100];  
 int age;  
 double heigh;  
 struct ListNode \*link;  
} ListNode;

# 15. 단순 연결 리스트가 정렬되지 않는 정수들의 리스트를 저장하고 있다. 리스트에서 최대값과 최소값을 찾는 프로그램을 작성하라.

int findMax(ListNode \*head) {  
 int max = head->data;  
 for (ListNode \*p = head; p != NULL; p = p->link) {  
 if (max < p->data) max = p->data;  
 }  
 return max;  
}  
  
int findMin(ListNode \*head) {  
 int min = head->data;  
 for (ListNode \*p = head; p != NULL; p = p->link) {  
 if (min > p->data) min = p->data;  
 }  
 return min;  
}

# 16. 단순 연결 리스트의 헤드 포인터가 주어져 있을 때 첫 번째 노드에서부터 하나씩 건너서 있는 노드를 전부 삭제하는 함수를 작성하라. 즉 홀수 번째 있는 노드들이 전부 삭제된다.

void deleteOddNode(ListNode \*head) {  
 if (head == NULL) return;  
  
 ListNode \*p = head;  
 while ((p != NULL) && (p->link != NULL)) {  
 ListNode \*removed = p->link;  
 p->link = removed->link;  
 free(removed);  
  
 p = p->link;  
 }  
}

# 17. 두 개의 단순 연결 리스트 A, B가 주어져 있을 경우, alternate 함수를 작성하라. alternate 함수는 A와 B로부터 노드를 번갈아 가져와서 새로운 리스트 C를 만드는 연산이다. 만약 입력 리스트 중에서 하나가 끝나게 되면 나머지 노드들을 전부 C로 옮긴다. 함수를 구현하여 올바르게 동작하는지 테스트하라. 작성된 함수의 시간 복잡도를 구하라.

ListNode\* alternate(ListNode \*head1, ListNode \*head2) {  
 ListNode \*a = head1;  
 ListNode \*b = head2;  
 ListNode \*c = NULL;  
  
 c = insert\_first(c, a->data);  
 ListNode \*tail = c->link;  
 while ((a != NULL) && (b != NULL)) {  
 if (b != NULL) {  
 c = insert(c, tail, b->data);  
 tail = tail->link;  
 b = b->link;  
 }  
 if (a != NULL) {  
 c = insert(c, tail, a->data);  
 tail = tail->link;  
 a = a->link;  
 }  
 }  
 return c;  
}

alternate 함수의 시간 복잡도: a와 b가 각 연결 리스트의 크기라고 가정했을 시, O(a+b)

# 18. 2개의 단순 연결 리스트를 병합하는 함수를 조금 변경하여 보자. 두 개의 연결 리스트 *a*=(*a*1, *a*2, …, *a*n), *b*=(*b*1, *b*2, …, *b*n)가 데이터 값의 오름차순으로 노드들이 정렬되어 있는 경우, 이러한 정렬 상태를 유지하면서 합병을 하여 새로운 연결 리스트를 만드는 알고리즘 merge를 작성하라. *a*와 *b*에 있는 노드들은 전부 새로운 연결 리스트로 옮겨진다. 작성된 알고리즘의 시간 복잡도도 구하라.

『===================(단순 연결 리스트 알고리즘) ===================』

# 19. 단순 연결 리스트 C를 두 개의 단순 연결 리스트 A와 B로 분리하는 함수 split를 작성하여 보자. C의 홀수 번째 노드들은 모두 A로 이동되고 C의 짝수 번째 노드들은 모두 B로 이동된다. 이 함수가 C를 변경하여서는 안 된다. 작성된 알고리즘의 시간 복잡도를 구하고 구현하여 수행하여 보라.

『===================(단순 연결 리스트 알고리즘) ===================』

# 20. 두 개의 다항식이 다음과 같이 주어졌다. 이들을 연결 리스트를 이용하여 나타내고 본문의 프로그램을 이용하여 두 다항식의 합을 구해보시오. *A*(*x*)=3*x*6+7*x*3-2*x*2-9, *B*(*x*)=-2*x*6-4*x*4+6*x*2+6*x*+1

『===================(단순 연결 리스트 알고리즘) ===================』

# 21. 다항식을 연결 리스트로 표현할 수 있음을 보였다. 다항식이 연결 리스트로 표현되어 있고, p를 다항식을 가리키는 포인터라고 할 때, 어떤 실수 x에 대하여 이 다항식의 값을 계산하는 함수 poly\_eval을 작성하라. 즉 다항식이 x3+2x+6이고 x=2이면 23+2·2+6를 계산하는 함수를 작성하여 보라.

『===================(단순 연결 리스트 알고리즘) ===================』

# 22. 배열을 이용하여 숫자들을 입력 받아 항상 오름차순으로 정렬된 상태로 유지하는 리스트 SortedList를 구현하여 보라. 다음의 연산들을 구현하면 된다.

『===================(단순 연결 리스트 알고리즘) ===================』

# 23. 단순 연결 리스트를 이용하여 숫자들을 항상 오름차순으로 정렬된 상태로 유지하는 리스트 SortedList를 구현하여 보라. 앞의 문제의 연산들을 구현하면 된다.

『===================(단순 연결 리스트 알고리즘) ===================』

# 24. 행렬(matrix)은 숫자나 문자를 정사각형 또는 직사각형으로 배열하여 그 양끝을 괄호로 묶은 것으로 많은 문제를 수학적으로 해결하는 도구이다. 희소 행렬(sparse matrix)은 많은 항들이 0인 행렬이다. 연결 리스트를 이용하여 희소 행렬을 표현하는 방법을 생각하여 보고 구현하라.

『===================(단순 연결 리스트 알고리즘) ===================』