|  |  |  |  |
| --- | --- | --- | --- |
| 2022/2 『자료구조』과제 보고서 | | | |
| 제목 | 6장 실습( ) 과제( O ) | 제출일자 | 2022.  11 .    09 . |
| 학번 | 201911608 | 이름 | 김지환 |

|  |
| --- |
| 02. 리스트의 n번째 요소를 가장 빠르게 찾을 수 있는 구현 방법은 무엇인가?  (1) 배열 (2) 단순 연결 리스트 (3) 원형 연결 리스트 (4)이중 연결 리스트  -> 배열은 O(1) 나머지는 O(n)이다. |
| 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;  -> (a), (b) 는 증감 연산자, (d)는 p 구조체의 data 멤버 변수를 받아온다.  (c)는 p 구조체의 연결된 다음 노드를 방문한다. |
| 06. 다음과 같이 변수 q가 1을 저장하는 노드를 가리키도록 하는 문장을 작성하라.    #include <stdio.h>  #include <stdlib.h>  typedef struct {  int data;  struct ListNode\* link;  }ListNode;  ListNode\* insert\_first(ListNode\* head, int value) {  ListNode\* p = (ListNode\*)malloc(sizeof(ListNode));  p->data = value;  p->link = head;  head = p;  return head;  }  void print\_list(ListNode\* head) {  for (ListNode\* p = head;p != NULL; p = p->link)  printf("%d -> ", p->data);  printf("NULL \n");  }  ListNode\* pointer(ListNode\* head, int value) {  ListNode\* p;  for (p = head; p->data != 1; p = p->link);  return p;  }  int main() {  ListNode\* head = NULL;  head = insert\_first(head, 1);  head = insert\_first(head, 2);  head = insert\_first(head, 3);  print\_list(head);  ListNode\* q = pointer(head, 1);  printf("변수 q의 data = %d\n", q->data);  return 0;  }  실행결과 |
| 08. 덱(deque: double-ended queue)은 삽입과 삭제가 양 끝에서 임의로 수행되는 자료 구조이다. 다음 그림과 같이 단순 연결 리스트(singly linked list)로 덱을 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)    ⓵ insertFirst 연산 : 덱의 첫 번째 원소로 삽입  ⓶ insertLast 연산 : 덱의 마지막 원소로 삽입  ⓷ deleteFirst 연산 : 덱의 첫 번째 원소를 삭제  ⓸ deleteLast 연산 : 덱의 마지막 원소를 삭제  1 -> dq->data = data, dq->link = first, first = dq; O(1)  2 -> dq -> data, dq->link = NULL, fisrt->last = dq; O(1)  3 -> first = dq->link, data = dq->data, free(dq) return data; O(1)  4 -> dq->link == first->tail 일 때까지 찾아야하므로 O(n) |
| 10. 다음과 같이 단순 연결리스트의 노드들의 개수를 계산하는 프로그램을 작성해보자.    #define \_CRT\_SECURE\_NO\_WARNINGS  #include <stdio.h>  #include <stdlib.h>  typedef struct {  int data;  struct ListNode\* link;  }ListNode;  ListNode\* insert(ListNode\* head, int v) {  ListNode\* p = (ListNode\*)malloc(sizeof(ListNode));  p->data = v;  p->link = head;  head = p;  return head;  }  int getSize(ListNode\* head) {  int size = 0;  for (head, size; head != NULL; head = head->link, size++);  return size;  }  int main() {  int n, i, data;  ListNode\* head = NULL;  printf("노드의 개수 : ");  scanf("%d", &n);  for (i = 1;i <= n; i++) {  printf("노드 #%d 데이터 : ", i);  scanf("%d", &data);  head = insert(head, data);  }  printf("연결 리스트 노드의 개수 : %d", getSize(head));  }  실행결과 |
| 12. 연결 리스트에서 특정한 데이터 값을 갖는 노드의 개수를 계산하는 함수를 작성하라  .  #define \_CRT\_SECURE\_NO\_WARNINGS  #include <stdio.h>  #include <stdlib.h>  typedef struct {  int data;  struct ListNode\* link;  }ListNode;  ListNode\* insert(ListNode\* head, int v) {  ListNode\* p = (ListNode\*)malloc(sizeof(ListNode));  p->data = v;  p->link = head;  head = p;  return head;  }  int getSizeElement(ListNode\* head, int v) {  int size = 0;  for (ListNode\* p = head; p != NULL; p = p->link)  if (p->data == v) size++;  return size;  }  int main() {  int n, i, data;  ListNode\* head = NULL;  printf("노드의 개수 : ");  scanf("%d", &n);  for (i = 1;i <= n; i++) {  printf("노드 #%d 데이터 : ", i);  scanf("%d", &data);  head = insert(head, data);  }  printf("탐색할 값을 입력하시오: ");  scanf("%d", &n);  printf("%d는 연결리스트에서 %d번 나타납니다.\n", n, getSizeElement(head, n));  }  실행결과 |
| 14. 다음 그림과 같은 데이터를 저장할 수 있는 단순 연결 리스트를 생성하는 프로그램을 작성    #include <stdio.h>  #include <stdlib.h>  typedef struct {  char name[100];  int age;  float height;  }element;  typedef struct {  element data;  struct ListNode\* link;  }ListNode;  ListNode\* insert(ListNode\* head, element v) {  ListNode\* p = (ListNode\*)malloc(sizeof(ListNode));  p->data = v;  p->link = head;  return p;  }  element set(char\* name, int age, float height) {  element data;  strcpy(data.name, name);  data.age = age;  data.height = height;  return data;  }  void print\_list(ListNode\* head) {  printf(" -------HEAD-------\n");  for (ListNode\* p = head; p != NULL; p = p->link) {  printf("\tName : %s\n\tAge : %d\n\theight : %.1f\n", p->data.name, p->data.age, p->data.height);  if(p->link!=NULL) printf(" ------------------\n");  }  printf(" -------NULL-------\n");  }  int main() {  ListNode\* head = NULL;  element data;  data = set("kim", 34, 1.7);  head = insert(head, data);  data = set("park", 27, 1.2);  head = insert(head, data);  data = set("lee", 48, 1.4);  head = insert(head, data);  data = set("choi", 30, 1.3);  head = insert(head, data);  print\_list(head);  return 0;  }  실행결과 |
| 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.  #include <stdio.h>  #include <stdlib.h>  typedef struct {  int data;  struct ListNode\* link;  }ListNode;  ListNode\* insert(ListNode\* head, int v) {  ListNode\* p = (ListNode\*)malloc(sizeof(ListNode));  p->data = v;  p->link = head;  return p;  }  void print\_list(ListNode\* head) {  for (ListNode\* p = head;p != NULL;p = p->link) {  printf("%d ", p->data);  if (p->link != NULL) printf("-> ");  }  printf("\n");  }  ListNode\* delete(ListNode\* head, ListNode\* pre) {  ListNode\* removed;  removed = pre->link;  if (removed == NULL) pre->link = NULL;  else {  pre->link = removed->link;  free(removed);  }  return head;  }  ListNode\* delete\_first(ListNode\* head) {  ListNode\* removed;  if (head == NULL) return NULL;  removed = head;  head = removed->link;  free(removed);  return head;  }  ListNode\* plop(ListNode\* head) {  int i = 0;  head = delete\_first(head);  ListNode\* p = head;  ListNode\* q;  for (p, i; p != NULL; p = p->link, i++) {  q = p->link;  head = delete(head, p);  }  return head;  }  int main() {  int i;  ListNode\* head = NULL;  for (i = 1;i <= 10;i++) head = insert(head, i);  print\_list(head);  printf("=======plop 수행========\n");  head = plop(head);  print\_list(head);  return 0;  }  실행결과 |
| 18. 2개의 단순 연결 리스트를 병합하는 함수를 조금 변경하여 보자. 두 개의 연결리스트 a=(a1,a2, ... , an), b = (b1, b2, ..., b3)가 데이터 값의 오름차순으로 노드들이 정렬되어 있는 경우, 이러한 정렬 상태를 유지하면서 합병을 하여 새로운 연결리스트를 만드는 알고리즘 merge를 작성하라. a와 b에 있는 노드들은 전부 새로운 연결 리스트로 옮겨진다. 작성된 알고리즘의 시간복잡도도 구하라.  #include <stdio.h>  #include <stdlib.h>  typedef struct {  int data;  struct ListNode\* link;  }ListNode;  ListNode\* insert(ListNode\* head, int v) {  ListNode\* p = (ListNode\*)malloc(sizeof(ListNode));  p->data = v;  p->link = head;  return p;  }  void print\_list(ListNode\* head) {  for (ListNode\* p = head; p != NULL; p = p->link) {  printf("%d ", p->data);  if (p->link != NULL) printf("-> ");  }  printf("\n");  }  int getSize(ListNode\* head) {  ListNode\* p = head;  int size = 0;  for (p, size; p != NULL; p = p->link, size++);  return size;  }  ListNode\* sort(ListNode\* head) {  int i = 0, max\_num = 0, j;  ListNode\* p = head;  for (p, i; p != NULL; p = p->link, i++) {  max\_num = max(max\_num, p->data);  }  int\* counting = (int\*)malloc(sizeof(int)\*(max\_num+1));  for (i = 0;i <= max\_num;i++) counting[i] = 0; // n  for (p=head; p != NULL; p = p->link) counting[p->data]++; // n  p = NULL;  for (i = max\_num; i >= 0; i--)  for (j = 0; j < counting[i];j++)  p = insert(p, i);  free(counting);  return p;  }  ListNode\* merge(ListNode\* head1, ListNode\* head2) {  if (head1 == NULL) return head2;  if (head2 == NULL) return head1;  ListNode\* p = head1;  while (p->link != NULL) p = p->link;  p->link = head2;  return sort(head1);  }  int main() {  ListNode\* A = NULL, \* B = NULL;  srand(time(NULL));  printf("==== 랜덤 리스트 A, B 생성 ====\n");  for (int i = 0;i < 5;i++) A = insert(A, rand() % 10 + 1);  printf("List A : "); print\_list(A);  for (int i = 0;i < 5;i++) B = insert(B, rand() % 10 + 1);  printf("List B : "); print\_list(B);  printf("==== 랜덤 리스트 A, B 오름차순 정렬 ====\n");  A = sort(A); B = sort(B);  printf("Sorted List A : "); print\_list(A);  printf("Sorted List B : "); print\_list(B);  printf("==== 정렬된 리스트 A, B 병합 ====\n");  A = merge(A, B);  printf("Merged List A : "); print\_list(A);  return 0;  }  실행결과    Merge() ADT 시간복잡도  - list A의 요소 n만큼 while 이후 리스트 병합 -> n회 연산  병합시 n만큼 외 반환 연산 3회, 치환 연산 2회 -> 5회 연산  반환 연산 중 sort() 함수 실행  sort() ADT시간 복잡도  max값 구하기 -> n회 연산  카운팅 정렬 -> n회 연산  그 외 반환, 동적해제, 치환 연산 -> 7회 연산  총 n+4+(n+n+7) 회 연산 => 3n + 11 ==> 알고리즘의 시간복잡도 == O(n) |
| 20. 두 개의 다항식이 다음과 같이 주어졌다. 이들을 연결 리스트를 이용해 나타내고 본문의 프로그램을 이용해 두 다항식의 합을 구해보시오.  A(x) = 3x^6 + 7x^3 - 2x^2 - 9  B(x) = -2x^6 - 4x^4 + 6x^2 + 6x + 1  #include <stdio.h>  #include <stdlib.h>  typedef struct ListNode {  int coef;  int expon;  struct ListNode\* link;  } ListNode;  typedef struct ListType {  int size;  ListNode\* head;  ListNode\* tail;  } ListType;  void error(char\* message) {  fprintf(stderr, "%s\n", message);  exit(1);  }  ListType\* create() {  ListType\* plist = (ListType\*)malloc(sizeof(ListType));  plist->size = 0;  plist->head = plist->tail = NULL;  return plist;  }  void insert\_last(ListType\* plist, int coef, int expon) {  ListNode\* temp = (ListNode\*)malloc(sizeof(ListNode));  if (temp == NULL) error("메모리 할당 에러");  temp->coef = coef;  temp->expon = expon;  temp->link = NULL;  if (plist->tail == NULL) {  plist->head = plist->tail = temp;  }  else {  plist->tail->link = temp;  plist->tail = temp;  }  plist->size++;  }  void poly\_add(ListType\* plist1, ListType\* plist2, ListType\* plist3) {  ListNode\* a = plist1->head;  ListNode\* b = plist2->head;  int sum;  while (a && b) {  if (a->expon == b->expon) {  sum = a->coef + b->coef;  if (sum != 0) insert\_last(plist3, sum, a->expon);  a = a->link; b = b->link;  }  else if (a->expon > b->expon) {  insert\_last(plist3, a->coef, a->expon);  a = a->link;  }  else {  insert\_last(plist3, b->coef, b->expon);  b = b->link;  }  }  for (; a != NULL; a = a->link) insert\_last(plist3, a->coef, a->expon);  for (; b != NULL; b = b->link) insert\_last(plist3, b->coef, b->expon);  }  void poly\_print(ListType\* plist, char \*s) {  ListNode\* p = plist->head;  printf("%s = ", s);  for (; p; p = p->link) {  printf("%d^%d", p->coef, p->expon);  if (p->link != NULL) printf(" + ");  }  printf("\n");  }  int main(void) {  ListType\* list1, \* list2, \* list3;  list1 = create();  list2 = create();  list3 = create();  insert\_last(list1, 3, 6);  insert\_last(list1, 7, 3);  insert\_last(list1, -2, 2);  insert\_last(list1, -9, 0);  insert\_last(list2, -2, 6);  insert\_last(list2, -4, 4);  insert\_last(list2, 6, 2);  insert\_last(list2, 6, 1);  insert\_last(list2, 1, 0);  poly\_print(list1, "A(x)");  poly\_print(list2, "B(x)");  poly\_add(list1, list2, list3);  poly\_print(list3, "A(x)+B(x)");  free(list1); free(list2); free(list3);  }  실행결과 |