|  |  |  |  |
| --- | --- | --- | --- |
| 2022/2 『자료구조』실습 보고서 | | | |
| 제목 | 8장 실습( O ) 과제( ) | 제출일자 | 2022.  11 .    17 . |
| 학번 | 201911608 | 이름 | 김 지환 |

|  |
| --- |
| 1. 프로그램 8.8 - 스레드 이진 트리 순회 프로그램  #include <stdio.h>  #define TRUE 1  #define FALSE 0  typedef struct TreeNode {  int data;  struct TreeNode\* left, \* right;  int is\_thread;  } TreeNode;  TreeNode n1 = { 'A', NULL, NULL, 1 };  TreeNode n2 = { 'B', NULL, NULL, 1 };  TreeNode n3 = { 'C', &n1, &n2, 0 };  TreeNode n4 = { 'D', NULL, NULL, 1 };  TreeNode n5 = { 'E', NULL, NULL, 0 };  TreeNode n6 = { 'F', &n4, &n5, 0 };  TreeNode n7 = { 'G', &n3, &n6, 0 };  TreeNode\* exp = &n7;  TreeNode\* find\_successor(TreeNode\* p) {  TreeNode\* q = p->right;  if (q == NULL || p->is\_thread == TRUE) return q;  while (q->left != NULL) q = q->left;  return q;  }  void thread\_inorder(TreeNode\* t) {  TreeNode\* q;  q = t;  while (q->left) q = q->left;  do {  printf("%c -> ", q->data);  q = find\_successor(q);  } while (q);  }  int main(void)  {  n1.right = &n3;  n2.right = &n7;  n4.right = &n6;  thread\_inorder(exp);  printf("\n");  return 0;  }  실행결과    ->    -> 검은 줄은 기존 노드간의 간선, 빨간 줄은 multi thred이다.  -> 모든 노드를 순차적으로 탐색하기 위해 가장 왼쪽 혹은 가장 오른쪽부터 트리를 탐색한다. 하지만 쓰레드가 right 노드로 연결되어 있으므로 왼쪽부터 탐색한다.  1. 가장 왼쪽 노드로 이동해 A 노드를 종료하고 쓰레드 내 프로세싱 중인 C 노드로 이동  2. C 노드를 종료하고 다음 프로세스인 B노드로 이동  3. B 노드를 종료하고 쓰레드 내 프로세싱 중인 G노드로 이동  4. G 노드를 종료하고 다음 프로세스인 D노드로 이동  5. D 노드를 종료하고 쓰레드 내 프로세싱 중인 F노드로 이동  6. F 노드를 종료하고 E노드로 이동 후 E노드도 종료. |
| 2. 프로그램 8.13 - 이진 탐색 트리의 중위순회 결과  #include <stdio.h>  #include <stdlib.h>  typedef int element;  typedef struct TreeNode {  element key;  struct TreeNode\* left, \* right;  } TreeNode;  typedef struct {  TreeNode\* data[10];  int front, rear;  } QueueType;  void init\_queue(QueueType\* q) {  q->front = q->rear = 0;  }  void eq(QueueType\* q, TreeNode\* item) {  if ((q->rear + 1) % 10 == q->front) return;  q->rear = (q->rear + 1) % 10;  q->data[q->rear] = item;  }  TreeNode\* dq(QueueType\* q) {  q->front = (q->front + 1) % 10;  return q->data[q->front];  }  TreeNode\* search(TreeNode\* node, int key) {  if (node == NULL) return NULL;  if (key == node->key) return node;  else if (key < node->key)  return search(node->left, key);  else  return search(node->right, key);  }  TreeNode\* new\_node(int item) {  TreeNode\* temp = (TreeNode\*)malloc(sizeof(TreeNode));  temp->key = item;  temp->left = temp->right = NULL;  return temp;  }  TreeNode\* insert\_node(TreeNode\* node, int key) {  if (node == NULL) return new\_node(key);  if (key < node->key)  node->left = insert\_node(node->left, key);  else if (key > node->key)  node->right = insert\_node(node->right, key);  return node;  }  TreeNode\* min\_value\_node(TreeNode\* node) {  TreeNode\* current = node;  while (current->left != NULL)  current = current->left;  return current;  }  TreeNode\* delete\_node(TreeNode\* root, int key) {  if (root == NULL) return root;  if (key < root->key)  root->left = delete\_node(root->left, key);  else if (key > root->key)  root->right = delete\_node(root->right, key);  else {  if (root->left == NULL) {  TreeNode\* temp = root->right;  free(root);  return temp;  }  else if (root->right == NULL) {  TreeNode\* temp = root->left;  free(root);  return temp;  }  TreeNode\* temp = min\_value\_node(root->right);  root->key = temp->key;  root->right = delete\_node(root->right, temp->key);  }  return root;  }  void inorder(TreeNode\* root) {  if (root) {  inorder(root->left);  printf("[%d] ", root->key);  inorder(root->right);  }  }  void print\_node(TreeNode\* root) {  QueueType q;  init\_queue(&q);  eq(&q, root);    while (q.rear != q.front) {  TreeNode\* data = dq(&q);  if(data->left != NULL) eq(&q, data->left);  if(data->right != NULL) eq(&q, data->right);  printf("%d ", data->key);  }  }  int main(void) {  TreeNode\* root = NULL;  root = insert\_node(root, 30);  root = insert\_node(root, 20);  root = insert\_node(root, 10);  root = insert\_node(root, 40);  root = insert\_node(root, 50);  root = insert\_node(root, 60);  print\_node(root);  printf("\n");  printf("이진 탐색 트리 중위 순회 결과 \n");  inorder(root);  printf("\n\n");  if (search(root, 30) != NULL)  printf("이진 탐색 트리에서 30을 발견함 \n");  else  printf("이진 탐색 트리에서 30을 발견못함 \n");  return 0;  }  실행결과    -> 맨 위의 출력은 data를 배열로 입력받은 결과이다.  -> print\_node 함수는 배열 순서대로 출력하기 위해 queue를 사용.  Insert 순서    배열 [30][20][40][10][ ][ ][50][ ][ ][ ][ ][ ][ ][60][ ]  해당 코드로 트리는 자동으로 오름차순으로 정렬됨을 알 수 있고 중위순회를 하게 되면  왼쪽 자식 -> 부모 -> 오른쪽 자식 순서로 출력함을 알 수 있으므로 오름차순대로 출력하게 된다.  아래는 예시로 25를 추가했을 때 결과 값을 출력한 것이다.    배열 [30][20][40][10][25][ ][50][ ][ ][ ][ ][ ][ ][60][ ] |
| 프로그램 8.14 - 영어사전 프로그램  #include <stdio.h>  #include <stdlib.h>  #include <string.h>  #include <memory.h>  #define MAX\_WORD\_SIZE 100  #define MAX\_MEANING\_SIZE 200  typedef struct {  char word[MAX\_WORD\_SIZE];  char meaning[MAX\_MEANING\_SIZE];  } element;  typedef struct TreeNode {  element key;  struct TreeNode \*left, \*right;  } TreeNode;  int compare(element e1, element e2) {  return strcmp(e1.word, e2.word);  }  void display(TreeNode \* p) {  if (p != NULL) {  printf("(");  display(p->left);  printf("%s:%s", p->key.word, p->key.meaning);  display(p->right);  printf(")");  }  }  TreeNode \* search(TreeNode \* root, element key) {  TreeNode \* p = root;  while (p != NULL) {  if (compare(key, p->key) == 0)  return p;  else if (compare(key, p->key) < 0)  p = p->left;  else if (compare(key, p->key) > 0)  p = p->right;  }  return p;  }  TreeNode \* new\_node(element item) {  TreeNode \* temp = (TreeNode \*)malloc(sizeof(TreeNode));  temp->key = item;  temp->left = temp->right = NULL;  return temp;  }  TreeNode \* insert\_node(TreeNode \* node, element key) {  if (node == NULL) return new\_node(key);  if (compare(key, node->key)<0)  node->left = insert\_node(node->left, key);  else if (compare(key, node->key)>0)  node->right = insert\_node(node->right, key);  return node;  }  TreeNode \* min\_value\_node(TreeNode \* node) {  TreeNode \* current = node;  while (current->left != NULL)  current = current->left;  return current;  }  TreeNode \* delete\_node(TreeNode \* root, element key) {  if (root == NULL) return root;    if (compare(key, root->key)<0)  root->left = delete\_node(root->left, key);    if (compare(key, root->key)>0)  root->right = delete\_node(root->right, key);  else {  if (root->left == NULL) {  TreeNode \* temp = root->right;  free(root);  return temp;  }  else if (root->right == NULL) {  TreeNode \* temp = root->left;  free(root);  return temp;  }  TreeNode \* temp = min\_value\_node(root->right);  root->key = temp->key;  root->right = delete\_node(root->right, temp->key);  }  return root;  }  void help() {  printf("\n\*\*\*\* i: 입력, d: 삭제, s: 탐색, p: 출력, q: 종료 \*\*\*\*: ");  }  int main(void) {  char command;  element e;  TreeNode \* root = NULL;  TreeNode \* tmp;  do {  help();  command = getchar();  getchar();  switch (command) {  case 'i':  printf("단어:");  gets(e.word);  printf("의미:");  gets(e.meaning);  root = insert\_node(root, e);  break;  case 'd':  printf("단어:");  gets(e.word);  root=delete\_node(root, e);  break;  case 'p':  display(root);  printf("\n");  break;  case 's':  printf("단어:");  gets(e.word);  tmp = search(root, e);  if (tmp != NULL)  printf("의미:%s\n", e.meaning);  break;  }  } while (command != 'q');  return 0;  }  실행결과    ->  8-13 이진 트리에서 문자열로 바뀐 결과이다.  삽입  cmp를 이용해서 문자열의 사전적 순서를 비교하고 사전적 순서로 앞일 경우 left 뒤일 경우 right 노드로 값을 삽입한다.  삭제  Cmp를 이용해서 사전적 순서로 입력받은 문자를 찾고 해당 문자가 존재하면 동적할당을 해제 후 삭제시킨다.  탐색  Cmp를 이용해서 사전적 순서로 해당 문자가 존재하면 출력 그렇지 않으면 null을 만날 때까지 반복해 null을 return한다. |
| 연습문제 13 – 균형 트리(좌우 서브트리 높이가 최대 1차이나는 이진 트리) 검사  - 이진 트리의 서브 트리 높이가 최대 1 차이나는 트리를 “균형 트리(balanced tree)”라고 한다. 주어진 이진 트리가 균형 트리인지를 검사하는 함수 isBalanced()를 작성하고 테스트하라.  #include <stdio.h>  #include <stdlib.h>  #define max(a,b) (((a)>(b))?(a):(b))  #define min(a,b) (((a)<(b))?(a):(b))  #define True 1  #define False 0  typedef int element;  typedef struct TreeNode {  element key;  struct TreeNode\* left, \* right;  } TreeNode;  TreeNode \* new\_node(element item) {  TreeNode \* temp = (TreeNode \*)malloc(sizeof(TreeNode));  temp->key = item;  temp->left = temp->right = NULL;  return temp;  }  TreeNode\* insert\_node(TreeNode\* node, int key) {  if (node == NULL) return new\_node(key);  if (key < node->key)  node->left = insert\_node(node->left, key);  else if (key > node->key)  node->right = insert\_node(node->right, key);  return node;  }  int get\_height(TreeNode\* root) {  int height = 0;  if(root != NULL)  height = 1+max(get\_height(root->left), get\_height(root->right));  return height;  }  int isBalanced(TreeNode\* root){  int l\_h = get\_height(root->left);  int r\_h = get\_height(root->right);    if(max(l\_h, r\_h) - min(l\_h, r\_h)>1) return False;  return True;  }  int main(void) {  TreeNode\* root = NULL;  root = insert\_node(root, 30);  root = insert\_node(root, 20);  root = insert\_node(root, 10);  root = insert\_node(root, 40);  root = insert\_node(root, 50);  root = insert\_node(root, 60);  if(isBalanced(root)) printf("균형 트리입니다.\n");  else printf("균형 트리가 아닙니다.\n");  return 0;  }  실행결과  프로그램 8.13의 경우    프로그램 8.13에서 오른쪽 높이 1을 추가할 경우  root = insert\_node(root, 70);     * 루트 트리의 왼쪽 트리 높이 l\_h * 루트 트리의 오른쪽 트리 높이 r\_h * l\_h 와 r\_h 중 더 큰 값 – 더 작은 값 = 높이 차 * 높이 차가 1보다 크면 균형 트리가 아님. |