데이터구조론 실습 –
트리(Tree)

HANYANG UNIVERSITY

Linked tree.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <memory.h>

typedef struct TreeNode {
int data;
struct TreeNode *left, *right;
} TreeNode;

void main()
{
```

```c
printf("n1 child : %d  %d₩n",n1->left->data, n1->right->data);
}
```

n1: 부모노드
n2, n3 자식노드

다음과 같이 트리를 작성해보세요

Linked tree.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <memory.h>

typedef struct TreeNode {
int data;
struct TreeNode *left, *right;
} TreeNode;

void main()
{
TreeNode *n1, *n2, *n3;
n1 = (TreeNode *)malloc(sizeof(TreeNode));
n2 = (TreeNode *)malloc(sizeof(TreeNode));
n3 = (TreeNode *)malloc(sizeof(TreeNode));

n1->data = 10;
n1->left = n2;
n1->right = n3;
n2->data = 20;
n2->left = NULL;
n2->right = NULL;
n3->data = 30;
n3->left = NULL;
n3->right = NULL;

printf("n1 child : %d  %d₩n",n1->left->data, n1->right->data);
}
```

TreeOrder.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <memory.h>

typedef struct TreeNode {
int data;
struct TreeNode *left, *right;
} TreeNode;

TreeNode n1 = { 1, NULL, NULL };
TreeNode n2 = { 4, &n1, NULL };
TreeNode n3 = { 16, NULL, NULL };
TreeNode n4 = { 25, NULL, NULL };
TreeNode n5 = { 20, &n3, &n4 };
TreeNode n6 = { 15, &n2, &n5 };
TreeNode *root = &n6;

void main()
{
inorder(root);
printf("₩n");
preorder(root);
printf("₩n");
postorder(root);
printf("₩n");
}
```

```c
void inorder(TreeNode *root) {
if (root) {
```

왼쪽 먼저 방문
현재 root의 값을 출력
오른쪽 방문

```c
}
}

void preorder(TreeNode *root) {
if (root) {
```

현재 root 값을 먼저 출력
왼쪽 방문
오른쪽 방문

```c
}
}

void postorder(TreeNode *root) {
if (root) {
```

왼쪽 먼저 방문
오른쪽 방문
현재 root의 값을 출력

```c
}
}
```

TreeOrder.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <memory.h>

typedef struct TreeNode {
int data;
struct TreeNode *left, *right;
} TreeNode;

TreeNode n1 = { 1, NULL, NULL };
TreeNode n2 = { 4, &n1, NULL };
TreeNode n3 = { 16, NULL, NULL };
TreeNode n4 = { 25, NULL, NULL };
TreeNode n5 = { 20, &n3, &n4 };
TreeNode n6 = { 15, &n2, &n5 };
TreeNode *root = &n6;

void main()
{
inorder(root);
printf("₩n");
preorder(root);
printf("₩n");
postorder(root);
printf("₩n");
}
```

```c
void inorder(TreeNode *root) {
if (root) {
inorder(root->left);
printf("%d(in) ", root->data);
inorder(root->right);
}
}

void preorder(TreeNode *root) {
if (root) {
printf("%d(pre) ", root->data);
preorder(root->left);
preorder(root->right);
}
}

void postorder(TreeNode *root) {
if (root) {
postorder(root->left);
postorder(root->right);
printf("%d(post) ", root->data);
}
}
```

Levelorder.c

```c
#include<stdio.h>
#include<stdlib.h>

#define MAX_QUEUE_SIZE 100

typedef struct TreeNode {
            int data;
            struct TreeNode *left, *right;
}TreeNode;

typedef TreeNode *element;

typedef struct {
            element queue[MAX_QUEUE_SIZE];
            int front, rear;
}QueueType;

TreeNode n1 = { 1 , NULL, NULL };
TreeNode n2 = { 4 , &n1, NULL };
TreeNode n3 = { 16 , NULL, NULL };
TreeNode n4 = { 25 , NULL, NULL };
TreeNode n5 = { 20 , &n3, &n4 };
TreeNode n6 = { 15 , &n2, &n5 };
TreeNode *root = &n6;
```

Levelorder.c

```
void error(char *message) {
            fprintf(stderr, "%s₩n", message);
            exit(1);
}

void init(QueueType *q) {
            q->front = q->rear = 0;
}

int is_empty(QueueType *q) {
            return (q->front == q->rear);
}

int is_full(QueueType *q) {
            return ((q->rear + 1) % MAX_QUEUE_SIZE == q->front);
}

void enqueue(QueueType *q, element item) {
            if (is_full(q))
                        error("Q is fUll");
            q->rear = (q->rear + 1) % MAX_QUEUE_SIZE;
            q->queue[q->rear] = item;
}

element dequeue(QueueType *q) {
            if (is_empty(q))
                        error("Q is empty");
            q->front = (q->front + 1) % MAX_QUEUE_SIZE;
            return q->queue[q->front];
}
```

Levelorder.c

```
void display_everything(QueueType *q) {
            int i, len;
            len = (q->front) - (q->rear);
            i = (q->front) + 1;
            while (i<MAX_QUEUE_SIZE) {
                        printf("%c₩n", q->queue[i]);
                        i++;
            }
            i = 0;
            while (i<len) {
                        printf("%c₩n", q->queue[i]);
                        i++;
            }
}
```

Levelorder.c

```
void level_order(TreeNode *ptr) {

            QueueType q;
            init(&q);
            if (ptr == NULL) return;

            enqueue(&q, ptr);

            while (!is_empty(&q)) {
                        ptr = dequeue(&q);
                        printf("%d ", ptr->data);
                        if (ptr->left)
                                    enqueue(&q, ptr->left);
                        if (ptr->right)
                                    enqueue(&q, ptr->right);
            }
}
int main(void) {
            level_order(root);
            return 0;
}
```