```c
#include <stdio.h>
#include <stdlib.h>
//#include "tree.h"
#include "linkqueue.h"

bitree * tree_create() {
	data_t ch;
	bitree *r;

	scanf("%c", &ch);
	if (ch == '#')
		return NULL;

	if ((r = (bitree *)malloc(sizeof(bitree))) == NULL) {
		printf("malloc failed\n");
		return NULL;
	}
	r->data = ch;
	r->left  = tree_create();
	r->right = tree_create();
	return r;
}

void preorder(bitree * r) {
	if (r == NULL) {
		return;
	}
	printf("%c", r->data);
	preorder(r->left);
	preorder(r->right);
}

void inorder(bitree * r) {
	if (r == NULL) {
		return;
	}
	inorder(r->left);
	printf("%c", r->data);
	inorder(r->right);
}

void postorder(bitree * r) {
	if (r == NULL) {
		return;
	}
	postorder(r->left);
	postorder(r->right);
	printf("%c", r->data);
}

void layerorder(bitree * r) {
	linkqueue * lq;

	if ((lq = queue_create()) == NULL)
		return;

	if (r == NULL)
		return;
```

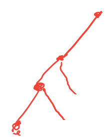
malloc → sizeof 接判

→ 初始化 → data left righ （除有效，有用 return free 还有核心的 延伸）

（scan ch 判）

~~（注意返回到哪，以及是什么的节点）~~

（注意到的哪个结点的哪个数位置）



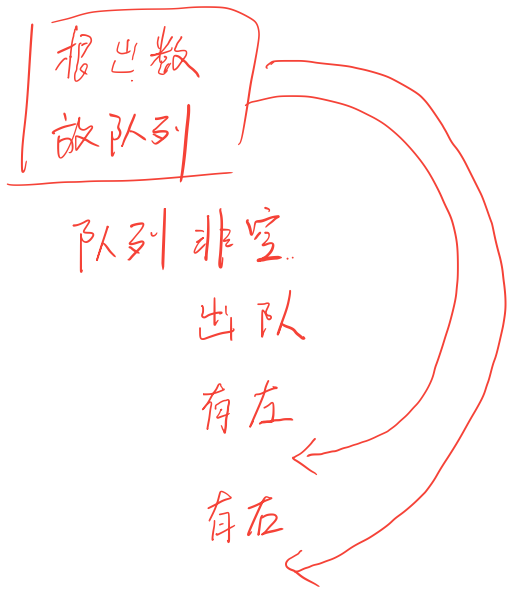
→ 打印数据 遍左结 遍右点 （r=NULL 结束）

→ 遍左 打印 遍右

→ 遍左 遍右 打印

```
        printf("%c", r->data);
        enqueue(lq, r);

        while (!queue_empty(lq)) {
                r = dequeue(lq);
                if (r->left) {
                        printf("%c", r->left->data);
                        enqueue(lq, r->left);
                }
                if (r->right) {
                        printf("%c", r->right->data);
                        enqueue(lq, r->right);
                }
        }
        puts("");
}
```



→ 循环(之前创队列，根入队始)

出队

左入队

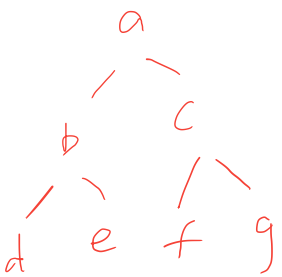
右入队

(然后左右也出入一次

然后因队顺序，回当，层次

遍出)

出后加印数



a b c # # e # # f # # g # #

(每到一个结点，都是一个函数，函数走完了算完)