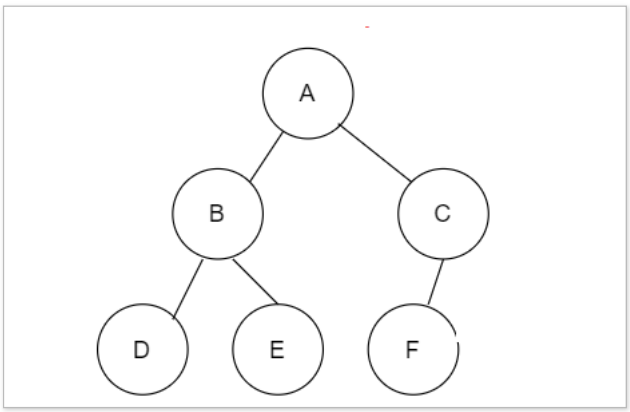
# 1.9 非递归遍历后序遍历_物联网 / 嵌入式工程师 - 慕课网

> 慕课网慕课教程 1.9 非递归遍历后序遍历涵盖海量编程基础技术教程，以图文图表的形式，把晦涩难懂的编程专业用语，以通俗易懂的方式呈现给用户。

9. 非递归遍历后序遍历

左右根, 遇根输出

对于后序，要保证根结点在左孩子和右孩子访问之后才能访问，因此对于任一结点 P，先将其入栈。如果 P 不存在左孩子和右孩子，则可以直接访问它；或者 P 存在左孩子或者右孩子，但是其左孩子和右孩子都已被访问过了，则同样可以直接访问该结点。若非上述两种情况，则将 P 的右孩子和左孩子依次入栈，这样就保证了每次取栈顶元素的时候，左孩子在右孩子前面被访问，左孩子和右孩子都在根结点前面被访问。




```
void post_order()
{
if(root == NULL)
        return ;

        linkstack_t *s = create_empty_linkstack();
        bitree_t *cur = NULL;
        bitree_t *pre = NULL;

        push_linkstack(s,root);

        while(!is_empty_linkstack(s))
        {
                cur = get_top_data(s);



                if((cur->lchild == NULL && cur->rchild == NULL) || \
                    (pre != NULL && (pre == cur->lchild || pre == cur->rchild)))
                {
                        printf("%c ",cur->data);
                        pop_linkstack(s);
                        pre = cur;
                }else{
                        if(cur->rchild != NULL)
                        {
                                push_linkstack(s,cur->rchild);
                        }

                        if(cur->lchild != NULL)
                        {
                                push_linkstack(s,cur->lchild);
```

```
                        }
                }
        }


        free(s);
    return ;
}
```

linkstack.h

```
#ifndef __LINKSTACK_H__
#define __LINKSTACK_H__

#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include "bitree.h"



typedef bitree_t *datatype_t;

typedef struct node
{
        datatype_t data;
        struct node *next;
}linknode_t;



typedef struct
{
        linknode_t *top;
        int n;
}linkstack_t;

extern linkstack_t *create_empty_linkstack();
extern int is_empty_linkstack(linkstack_t *s);
extern void push_linkstack(linkstack_t *s,datatype_t data);
extern datatype_t pop_linkstack(linkstack_t *s);
extern datatype_t get_top_data(linkstack_t *s);
#endif
```

linkstack.c

```
#include "linkstack.h"



linkstack_t *create_empty_linkstack()
{
        linkstack_t *s = NULL;

        s = (linkstack_t *)malloc(sizeof(linkstack_t));
        if(NULL == s)
        {
                printf("malloc is fail!\n");
                return NULL;
        }

        memset(s,0,sizeof(linkstack_t));

        return s;
}



int is_empty_linkstack(linkstack_t *s)
{
        return s->top == NULL ? 1 : 0;
}


void push_linkstack(linkstack_t *s,datatype_t data)
{
        linknode_t *temp = NULL;

        temp = (linknode_t *)malloc(sizeof(linknode_t));
        if(NULL == temp)
```

```
        {
                printf("malloc is fail!\n");
                return ;
        }

        temp->data = data;


        temp->next = s->top;
        s->top = temp;


        s->n ++;
        return ;
}


datatype_t pop_linkstack(linkstack_t *s)
{
        linknode_t *temp = NULL;
        datatype_t data;



        temp = s->top;


        data = temp->data;


        s->top = temp->next;


        free(temp);
        temp = NULL;


        s->n --;

        return data;
}


datatype_t get_top_data(linkstack_t *s)
{
        return s->top->data;
}
```

bitree.c

```
#include "bitree.h"
#include "linkstack.h"

bitree_t *create_binatry_tree(int n)
{
        bitree_t *root = NULL;

        root = (bitree_t *)malloc(sizeof(bitree_t));
        memset(root,0,sizeof(bitree_t));

        root->n = n;
        root->lchild = root->rchild = NULL;


        printf("Input %d node data : ",n);
        scanf("%c",&(root->data));


        while(getchar() != '\n');


        if(2 * n <= N)
        {
                root->lchild = create_binatry_tree(2 * n);
        }


        if(2 * n  + 1 <= N)
        {
                root->rchild = create_binatry_tree(2 * n + 1);
        }

        return root;
```

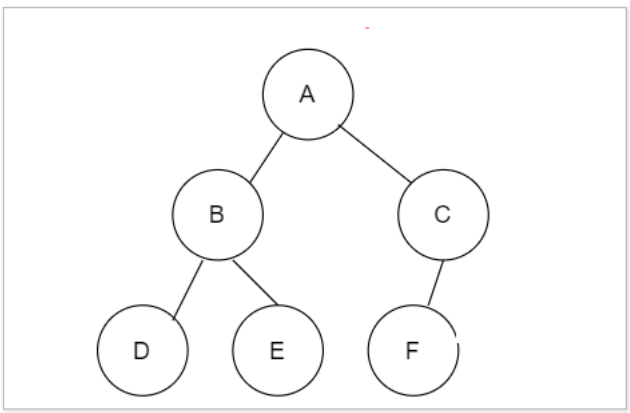
```
}


void pre_order(bitree_t *root)
{
        if(root == NULL)
                return ;

        linkstack_t *s = create_empty_linkstack();
        bitree_t *temp = root;

        while(temp != NULL || !is_empty_linkstack(s))
        {
                while(temp != NULL)
                {
                        printf("(%d : %c) ",temp->n,temp->data);
                        push_linkstack(s,temp);
                        temp = temp->lchild;
                }

                if(!is_empty_linkstack(s))
                {
                        temp = pop_linkstack(s);
                        temp = temp->rchild;
                }
        }

        free(s);
}


void in_order(bitree_t *root)
{

    if(root == NULL)
            return ;

    linkstack_t *s = create_empty_linkstack();

    bitree_t *node = root;

    while(node != NULL || !is_empty_linkstack(s))
    {

        if(node != NULL)
        {
            push_linkstack(s,node);
            node = node->lchild;
        }else{

            node = pop_linkstack(s);
            printf("%c ",node->data);
            node = node->rchild;
        }
    }

    free(s);
}


void post_order(bitree_t *root)
{
    if(root == NULL)
        return ;

    linkstack_t *s = create_empty_linkstack();
    bitree_t *cur = NULL;
    bitree_t *pre = NULL;

    push_linkstack(s,root);

    while(!is_empty_linkstack(s))
    {
            cur = get_top_data(s);

            if((cur->lchild == NULL && cur->rchild == NULL) || \
                            (pre != NULL && (pre == cur->lchild || pre == cur->rchild)))
            {
                    printf("%c ",cur->data);
                    pop_linkstack(s);
                    pre = cur;
            }else{
                    if(cur->rchild != NULL)
                    {
                            push_linkstack(s,cur->rchild);
                    }
```

```
                if(cur->lchild != NULL)
                {
                        push_linkstack(s,cur->lchild);
                }
            }
    }


    free(s);
    return ;
}
```

main.c

```
#include "bitree.h"









int main()
{
        bitree_t *root;

        root = create_binatry_tree(1);

        printf("create is successful!\n");

        printf("pre_order : ");
        pre_order(root);
        printf("\n");

        printf("in_order : ");
        in_order(root);
        printf("\n");

        printf("post_order : ");
        post_order(root);
        printf("\n");
        return 0;
}
```

写出下列图形层次遍历的代码。（答题区只需要提出贴出遍历的函数和运行结果即可）




解题思路：
二叉树的层次遍历：是指从二叉树的根结点开始，从上到下逐层遍历。在同一层中
按照从左到右的顺序对每一个结点进行访问。在进行层次遍历
的时候，当前层结点访问完成后，在依次对左孩子和右孩子进行一层
层访问，先遇到的结点先输出。这个和我们队列的思想（先进先出）比较符合。
因此，二叉树的层次遍历可以设置队列来操作。

二叉树层次遍历的操作：
1.先创建队列
2.层次遍历
(1)先把根结点入队
(2)当队列不为空的时候，循环出队，然后判断左孩子和右孩子是否存在，存在则入队.
当队列为空的时候，退出循环。

伪代码：

```
LinkQueue *q = creat_empyt_linkqueue();

enter_linqueue(root);
while(!is_emtpy_linkquue(q))
{
        temp = delte_linkqueu()
        printf("%c",temp->data)

        if(temp->lchild != NULL)
        {
             enter_linkqueue(q,temp->lchlid);
        }

        if(temp->rchild != NULL)
        {
             enter_linkqueue(q,temp->rchlid);
        }
}
```

全文完