1.8 非递归遍历中序遍历_物联网 / 嵌入式工程师 - 慕课网

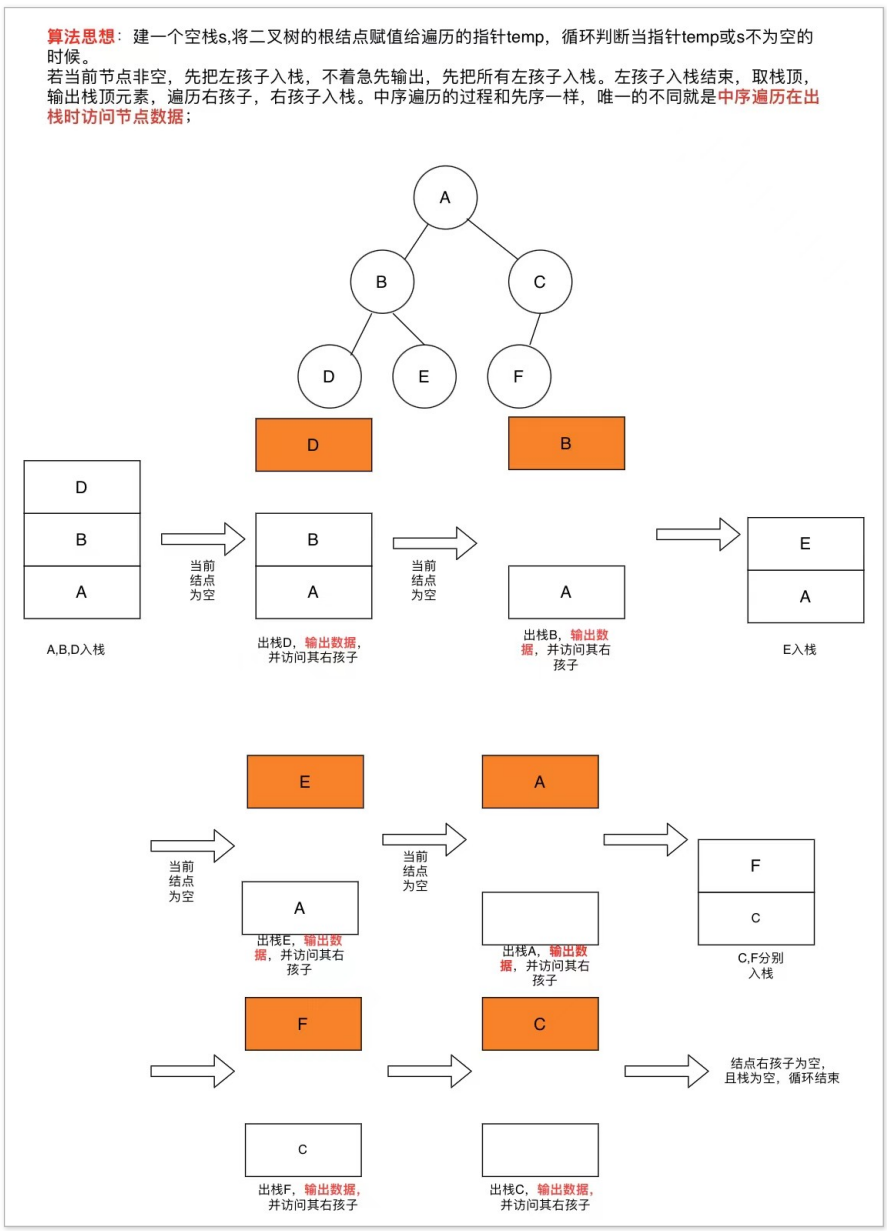
> 慕课网慕课教程 1.8 非递归遍历中序遍历涵盖海量编程基础技术教程，以图文图表的形式，把晦涩难懂的编程专业用语，以通俗易懂的方式呈现给用户。

8. 非递归遍历中序遍历

左根右, 遇根输出

创建一个空栈 s, 将二叉树的根结点赋值给遍历的指针 temp，循环判断当指针 temp 或 s 不为空的时候。

若当前节点非空，先把左孩子入栈，不着急先输出，先把所有左孩子入栈。左孩子入栈结束，取栈顶，输出栈顶元素，遍历右孩子，右孩子入栈。中序遍历的过程和先序一样，唯一的不同就是中序遍历在出栈时访问节点数据



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

    linkstack_t *s = create_empty_linkstack();

    bitree_t *temp = root;

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

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

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

    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.h

```
#ifndef __BITREE_H__
#define __BITREE_H__

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


#define N 6

typedef char data_t;

typedef struct bitree
```

```
{
        int n;
        data_t data;
        struct bitree *lchild;
        struct bitree *rchild;
}bitree_t;


extern bitree_t *create_binatry_tree(int n);
extern void pre_order(bitree_t *root);
extern void in_order(bitree_t *root);
extern void post_order(bitree_t *root);
#endif
```

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 *temp = root;

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

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

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

    free(s);
}


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

    post_order(root->lchild);
    post_order(root->rchild);
    printf("(%d : %c) ",root->n,root->data);
}
```

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;
}
```

全文完