栈概念：

从逻辑上对数据的处理遵循：后进先出/先进后出的逻辑就称为栈。

图 3-41 吃完饭洗洗碗

使用顺序存储 + 线性结构 + 栈的逻辑 = 顺序栈

链式存储 + 线性结构 + 栈的逻辑 = 链式栈

链式栈：

使用头插的方法来进行入栈（添加数据），头部移除数据（出栈）。

不管增加还是删除数据都在链表的同一端即可。

设计节点：

```
1
2  // 节点设计
3  typedef struct stack
4  {
5      struct book
6      {
7          char Name [32];
8          float Price ;
9          int Num ;
10     }Book;// 声明的时候顺便定义了一个变量名为 Book
11
12     struct stack * Next;
13
14 }Node , *P_Node;
```

```
1
```

```
typedef struct book
{
    char Name [32];
    float Price ;
    int Num ;
} Data_Type ;

// 节点设计
typedef struct stack
{
    Data_Type data ;
    struct stack * Next;

}Node , *P_Node;
```

初始化链式栈：

```
P_Node inint_new_node( Data_Type * data )
{
    P_Node new = calloc(1, sizeof( Node ));

    if( data != NULL ) // 判断是否不是头节点初始化， 则初始化节点的数据域
    {
        // 初始化书名
        strncpy(new->data.Name , data->Name , 32 );
        new->data.Num = data->Num ;
        new->data.Price = data->Price ;
    }

    new->Next = NULL ;

    return new ;
}
```

入栈：

```

```

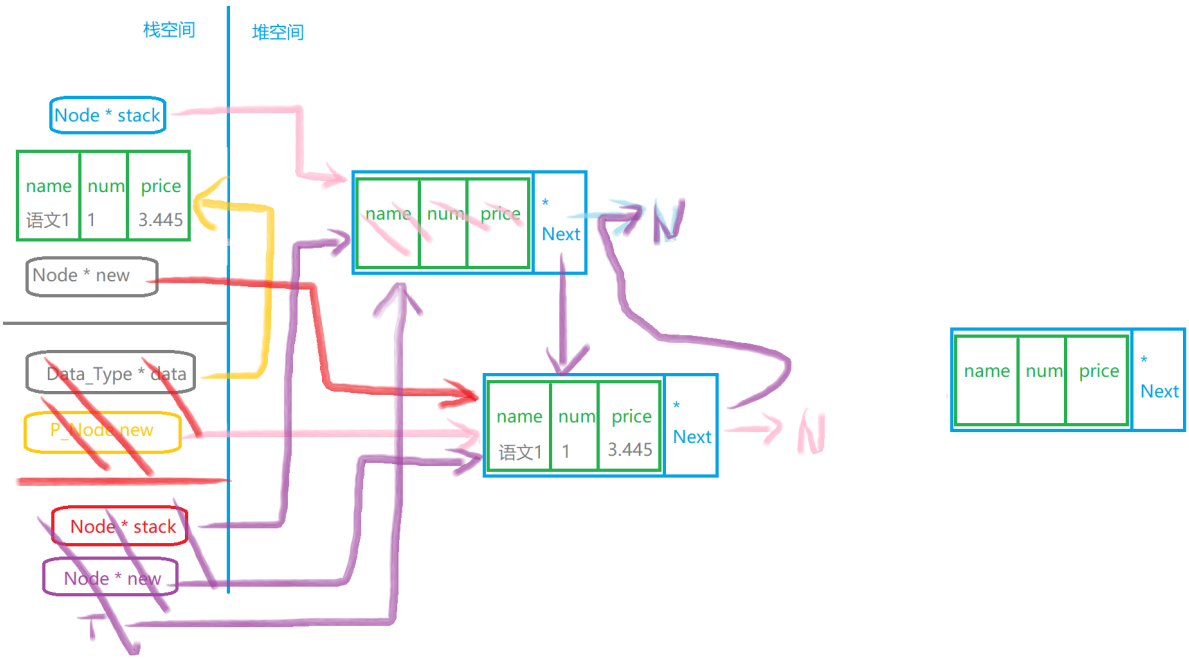
```
int push_stack( Node * stack , Node * new  )
{
    if(stack ==  NULL  || new == NULL )
    {
        return -1 ;
    }

    Node * tmp = stack ;

    new->Next = tmp->Next ;
    tmp->Next = new ;

    return 0 ;
}
```

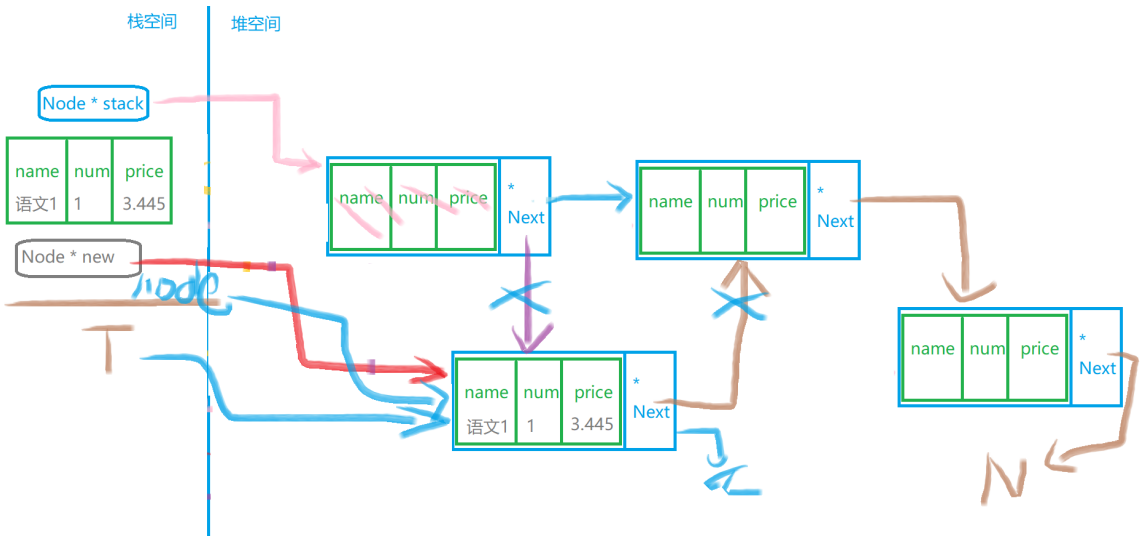
出栈

```

Node * pop_stack(Node * stack)
{
    if( stack->Next == NULL )
    {
        printf("当前栈为空！！\n");
        return NULL ;
    }

    Node * tmp =  stack->Next ;

    stack->Next = tmp->Next ;
    // stack->Next = stack->Next->Next ;

    tmp->Next = NULL ;

    return tmp ;
}
```

操作：

1. 读懂链式栈的代码以及逻辑
2. 尝试自己实现入栈/出栈的操作（用户输入数据）





顺序栈

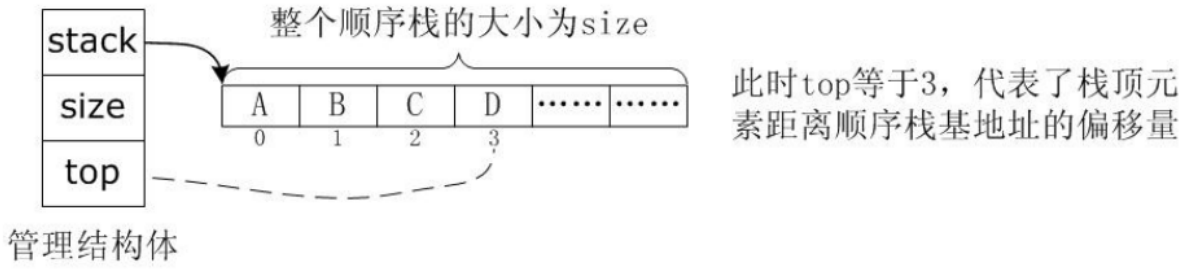


图 3-43 顺序栈

设计管理结构体：

```
struct node
{
    int * Enter ;   // 栈的入口地址（堆空间）
    int top;        // 栈顶元素的下标（偏移量）
    int size;       // 栈的大小（元素个数）
};
```

初始化：

```

Node stack_init(int size_of_stack )
{
    Node stack;
    stack.Enter = calloc( size_of_stack , sizeof(int));
    stack.top = 0 ;
    stack.size = size_of_stack ;

    return stack ; // 返回结构体变量（值的返回）
}
```

入栈：

```

Node push_stack( Node stack , int num )
{
    if( stack.top >= stack.size-1 ) // 判断栈是否已满
```

```
    {
        printf("不好意思，奶茶没有奶了！！\n");
        return stack ;
    }

    printf("num:%d\n" , num );

    *(stack.Enter + stack.top) = num ;
    // stack.Enter[stack.top] = num ;
    printf("stack.Enter + stack.top:%d\n" , *(stack.Enter + stack.top));

    stack.top ++ ;

    return stack ;
}
```

出栈

```
Node pop_stack(Node stack)
{
    if( stack.top == 0 )
    {
        printf("栈为空！！！\n");
        return stack ;
    }

    stack.top -- ;
    int tmp = stack.Enter[stack.top] ;

    printf("%d", tmp ) ;

    return stack ;
}
```