# C#堆栈

一、定义

**堆栈(Stack)最明显的特征就是“先进后出”**，本质上讲堆栈也是一种线性结构，符合线性结构的基本特点：即每个节点有且只有一个前驱节点和一个后续节点。[二、定义抽象堆栈的接口IStack](IStack.cs)

interface IStack<T> {

//返回堆栈得实际元素个数

int Count();

//判断堆栈是否为空

bool IsEmpty();

//清空堆栈里得元素

void Clear();

//入栈:将元素压入堆栈中

void Push(T item);

//出栈:从堆栈顶取一个元素，并从堆栈中删除

T Pop();

//取堆栈顶部的元素(但不删除)

T Peek();

}

三、顺序堆栈（SeqStack）

//顺序堆栈

public class SeqStack<T> : IStack<T> {

private int size;

private T[] data;

private int top;

public SeqStack(int size) {

this.size = size;

data = new T[size];

top = -1;

}

#region 接口实现部分

public int Count() {

return top + 1;

}

public void Clear() {

top = -1;

}

public void Push(T item) {

if (IsFull()) {

*Console*.*WriteLine*("数据已满！");

return;

}

data[++top] = item;

}

public T Pop() {

T tmp = default(T);

if (top == -1) {

*Console*.*WriteLine*("数据为空！");

return tmp;

}

tmp = data[top];

top--;

return tmp;

}

public T Peek() {

T tmp = default(T);

if (top == -1) {

*Console*.*WriteLine*("数据为空！");

return tmp;

}

return data[top];

}

#endregion

public bool IsFull() {

return top == size - 1;

}

public override string ToString() {

*StringBuilder* sb = new *StringBuilder*();

for (int i = top; i >= 0; i++)

sb.*Append*(data[i] + ",");

return sb.*ToString*().Trim(',');

}

public bool IsEmpty() {

return top == -1;

}

}

三、链式堆栈（SeqStack）

class LinkStack<T> : IStack<T> {

private Node<T> top;

private int num;//结点个数

public Node<T> Top {

get { return top;}

set { top = *value*; }

}

public LinkStack() {

top = null;

num = 0;

}

public int Count() {

return num;

}

public void Clear() {

top = null;

num = 0;

}

public bool IsEmpty() {

if (top == null && num == 0) return true;

else return false;

}

public void Push(T item) {

Node<T> node = new Node<T>(item);

if (top == null) top = node;

else {

node.Next = top;

top = node;

}

num++;

}

public T Pop() {

if (IsEmpty()) return default(T);

Node<T> node = top;

top = top.Next;

num--;

return node.*Data*;

}

public T Peek() {

if (IsEmpty()) return default(T);

return top.*Data*;

}

public override string ToString() {

*StringBuilder* sb = new *StringBuilder*();

if (top != null) {

sb.*Append*(top.*Data*.ToString() + ",");

Node<T> node = top;

while (node.Next != null) {

sb.*Append*(node.Next.*Data*.ToString() + ",");

node = node.Next;

}

}

return sb.ToString().Trim(',');

}

}

public class Node<T> {

private T data;

private Node<T> next;

public Node(T data, Node<T> next) {

this.data = data;

this.next = next;

}

public Node(Node<T> next) {

this.next = next;

this.data = default(T);

}

public Node(T data) {

this.data = data;

this.next = null;

}

public Node() {

this.data = default(T);

this.next = null;

}

public T Data {

get { return this.data; }

set { this.data = *value*; }

}

public Node<T> Next {

get { return next; }

set { next = *value*; }

}

}