**《数据结构A》期末试卷**

班级： 软件2201 学号： 222959587 姓名： 刘悦阳

一、简答题

1、逻辑结构有哪两种划分方法？分别包括哪些数据结构？

逻辑结构主要分为线性结构和非线性结构。

1. 线性结构：数据元素之间存在一对一的相互关系。线性结构包括以下几种数据结构：

- 数组

- 链表

- 堆

- 栈

- 队列

2. 非线性结构：数据元素之间的关系更为复杂。非线性结构主要包括以下几种数据结构：

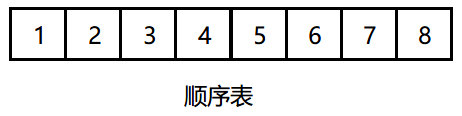
- 集合结构：数据元素之间只有同属于一个集合的关系，没有其他特定关系

- 树形结构：数据元素之间存在一对多的关系。例如，普通树、二叉树、二叉搜索树、平衡二叉搜索树、红黑树、B树、B+树等

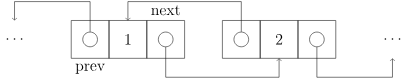
- 图形结构或网状结构：数据元素之间存在多对多的关系

2、画出下列数据结构的图示：（1）顺序表；（2）双向链表。

顺序表：



双向链表：



3、请简述顺序表和链表在进行插入操作时，有什么不同。

顺序表和链表在进行插入操作时的主要区别在于它们的数据结构和内存分配方式。

**顺序表**：

* 顺序表是一种线性表，它的元素在内存中是连续存储的。因此，插入操作需要移动大量元素。具体来说，如果在顺序表的第i个位置插入一个元素，那么需要移动第i个位置及其之后的所有元素。这样，顺序表的插入操作的时间复杂度为O(n)。
* 但是，如果在顺序表的末尾插入元素，那么这个操作的时间复杂度就是O(1)，因为不需要移动任何元素。

**链表**：

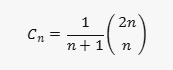
* 链表是一种线性表，但它的元素在内存中不是连续存储的。每个元素（或称为节点）包含两部分：一部分是存储数据的域，另一部分是指向下一个元素的指针。因此，插入操作只需要改变相应的指针，而不需要移动元素。具体来说，如果在链表的第i个位置插入一个元素，那么只需要改变第i-1个元素的指针和新插入的元素的指针。这样，链表的插入操作的时间复杂度为O(1)。
* 但是，需要注意的是，为了找到第i个位置，我们需要从链表的头部开始遍历，这个操作的时间复杂度是O(n)。

总的来说，顺序表和链表在插入操作上的主要区别是：顺序表插入元素需要移动元素，而链表插入元素只需要改变指针。因此，链表在插入操作上通常比顺序表更高效。但是，链表需要额外的空间来存储指针，而顺序表不需要。这是在选择使用顺序表还是链表时需要考虑的一个重要因素。

4、若依次读入数据元素序列“a、b、c、d”进栈，进栈的过程中允许出栈，请写出各种可能的出栈元素序列。

1. a、b、c、d
2. a、b、d、c
3. a、c、b、d
4. a、c、d、b
5. a、d、c、b
6. b、a、c、d
7. b、a、d、c
8. b、c、a、d
9. b、c、d、a
10. b、d、c、a
11. c、b、a、d
12. c、b、d、a
13. c、d、b、a
14. d、c、b、a

对于n个元素的序列，可能的出栈序列数是第n个Catalan数，计算公式为：



5、试写出算术运算式24-3\*4/6-5\*2的操作数栈和运算符栈的变化情况。

1. 从左到右扫描表达式。
2. 如果遇到操作数，我们就直接将其压入操作数栈。
3. 如果遇到运算符，我们就需要比较其与运算符栈顶运算符的优先级：
   * 如果运算符栈为空，或者栈顶运算符的优先级低于当前运算符，那么我们就将当前运算符压入运算符栈。
   * 如果栈顶运算符的优先级高于或等于当前运算符，那么我们就从操作数栈中取出两个操作数，从运算符栈中取出一个运算符，进行计算，然后将计算结果压入操作数栈，重复此步骤，直到运算符栈为空或者找到一个优先级低于当前运算符的运算符，然后将当前运算符压入运算符栈。
4. 重复步骤2和3，直到表达式的所有数字和操作符都被扫描处理过。
5. 如果扫描完表达式后，运算符栈中还有运算符，那么就从操作数栈中取出两个操作数，从运算符栈中取出一个运算符，进行计算，然后将计算结果压入操作数栈，重复此步骤，直到运算符栈为空。
6. 此时，操作数栈中只剩下一个数，这就是表达式的结果。

以下详细列出每一步的操作数栈和运算符栈的变化情况。这里假设乘法和除法的优先级高于加法和减法，且在同一优先级内，运算符遵循从左到右的顺序。

初始状态：

操作数栈：空

运算符栈：空

扫描到24：

操作数栈：24

运算符栈：空

扫描到-：

操作数栈：24

运算符栈：-

扫描到3：

操作数栈：24, 3

运算符栈：-

扫描到\*：

操作数栈：24, 3

运算符栈：-, \*

扫描到4：

操作数栈：24, 3, 4

运算符栈：-, \*

扫描到/：

操作数栈：24, 12 //执行一次乘法操作

运算符栈：-, /

扫描到6：

操作数栈：24, 12, 6

运算符栈：-, /

扫描到-：

操作数栈：24, 2 //执行一次除法操作

运算符栈：-, -

扫描到5：

操作数栈：24, 2, 5

运算符栈：-, -

扫描到\*：

操作数栈：22 //执行一次减法操作

运算符栈：\*

扫描到2：

操作数栈：22, 5, 2

运算符栈：\*

扫描结束：

操作数栈：22, 10 //执行一次乘法操作

运算符栈：空

最终结果：

操作数栈：12 //执行一次减法操作

运算符栈：空

所以，24-3\*4/6-5\*2 的结果是 12。操作数栈和运算符栈的变化情况如上所示。

二、解释代码含义

1、

public class StuNode{

private Student student;

private StuNode nextlink;

public Student getStudent(){

return student;

}

public void setStudent(Student student) {

this.student = student;

}

public StuNode getNext() {

return nextlink;

}

public void setNext(StuNodenextlink) {

this.nextlink = nextlink;

}

}

public class StuLinkList {

private StuNode head;

public StuLinkList(StuNodehd){

head = hd;

}

}

这段代码定义了两个类：StuNode 和 StuLinkList。

StuNode 类是一个节点类，它包含两个私有成员：student 和 nextlink。student 是 Student 类型的对象，nextlink 是 StuNode 类型的对象，表示下一个节点的链接。这个类提供了一些公共方法来获取和设置这两个成员的值：

* getStudent() 方法返回 student 对象。
* setStudent(Student student) 方法设置 student 对象。
* getNext() 方法返回下一个节点的链接 nextlink。
* setNext(StuNode nextlink) 方法设置下一个节点的链接 nextlink。

StuLinkList 类是一个链表类，它包含一个私有成员 head，表示链表的头节点。这个类有一个构造函数，接受一个 StuNode 类型的参数 hd，并将其设置为头节点。

总的来说，这段代码定义了一个单链表数据结构，其中每个节点包含一个 Student 对象和一个指向下一个节点的链接。链表类 StuLinkList 则用于管理这些节点。这种数据结构常用于实现队列、栈等数据结构，或者用于解决一些需要动态分配内存的问题。

2、

public class StackSLinked implements Stack {

private SLNode top;

private int size;

public StackSLinked() {

top = null;

size = 0;}

public int getSize() {

return size;

}

public boolean isEmpty() {

return size==0;

}

public void push(Object e) {

SLNode q= new SLNode(e,top);

top =q;

size++;

}

public Object pop() throws StackEmptyException {

if(size<1)

throw new StackEmptyException("错误，堆栈为空。");

Object obj = top.getData();

top=top.getNext();

size--;

return obj;

}

public Object peek() throws StackEmptyException {

if(size<1)

throw new StackEmptyException("错误，堆栈为空。");

return top.getData();

}

}

这段代码定义了一个基于链表实现的栈(StackSLinked)。以下是每个方法的解释：

* public StackSLinked(): 这是构造函数，用于初始化栈。它将栈顶(top)设置为null，并将栈的大小(size)设置为0。
* public int getSize(): 这个方法返回栈的大小(size)。
* public boolean isEmpty(): 这个方法检查栈是否为空。如果栈的大小(size)为0，那么栈就是空的，返回true；否则返回false。
* public void push(Object e): 这个方法用于将一个元素(e)压入栈顶。它首先创建一个新的节点(q)，然后将这个新节点设置为栈顶(top)，并将栈的大小(size)增加1。
* public Object pop() throws StackEmptyException: 这个方法用于从栈顶弹出一个元素。如果栈为空（即栈的大小(size)小于1），那么它会抛出一个StackEmptyException异常。否则，它会获取栈顶元素的数据(top.getData())，将栈顶设置为下一个节点(top.getNext())，并将栈的大小(size)减少1。最后，它返回弹出的元素。
* public Object peek() throws StackEmptyException: 这个方法用于查看（但不移除）栈顶的元素。如果栈为空（即栈的大小(size)小于1），那么它会抛出一个StackEmptyException异常。否则，它返回栈顶的元素。