# 明确目标——考试大纲

## 一、考试的基本要求

要求考生比较系统地掌握数据结构与程序设计的基本概念和理论，不仅能解答基础知识题，且能综合运用所学的基本数据结构和程序设计技术给出相应实际问题的解决方案，具备对实际问题进行分析，抽象出计算机加工的数据对象及其间的关系，能选择的合适的数据结构表达应用问题，并选择合适的存储结构实现相应的程序设计，解决实际应用问题。要求考生具有计算机专业学生的基本素质，具有良好的程序设计风格，能综合运用所学的基本数据结构和程序设计知识分析问题、解决问题。

## 二、考试方法和考试时间

数据结构与程序设计考试采用闭卷笔试形式，试卷满分150分，考试时间180分钟。

## 三、考试内容

* 1. 数据结构概述
     1. 数据结构的基本概念以及相关术语；
     2. 数据结构的抽象数据类型表示与实现；
     3. 算法的基本概念及设计要求；
     4. 算法的时间复杂度和空间复杂度分析。
  2. 线性表
     1. 线性表的类型定义、基本算法，以及集合运算等应用问题的解决方案；
     2. 线性表的顺序表示和实现，及集合运算等应用问题的解决方案的顺序表实现；
     3. 线性表的链式表示和实现，及集合运算等应用问题的解决方案的单链表实现；
     4. 一元多项式的表示及相加，即一元多项式的单链表加法算法实现。
  3. 栈和队列
     1. 栈的类型定义；
     2. 栈的顺序存储表示和栈的基本操作在顺序栈的实现；
     3. 栈的链式存储表示和实现；
     4. 队列的类型定义；
     5. 队列的链式存储表示和实现；
     6. 队列的顺序存储表示和实现，特别是循环队列。
  4. 串
     1. 串的类型定义；
     2. 串的表示和实现；
     3. 串的模式匹配算法。
  5. 数组和广义表
     1. 数组的定义；

2．数组的顺序表示和实现；

3．矩阵的压缩存储；

4．广义表的定义；

5．广义表的存储结构；

6．m元多项式的表示；

7．广义表的递归算法

* 1. 树和二叉树
     1. 树的定义和基本术语；
     2. 二叉树的定义及性质；
     3. 二叉树的二叉链表存储结构；
     4. 遍历二叉树和线索二叉树；
     5. 树的各种存储结构；
     6. 树、森林与二叉树的转换；
     7. 树和森林的遍历；
     8. 哈夫曼树及其应用；
     9. 回溯法与树的遍历；
     10. 树的计数。
  2. 图
     1. 图的类型定义；
     2. 图的存储结构；
     3. 图的遍历；
     4. 图的连通性问题；
     5. 拓扑排序及关键路径；
     6. 最短路径。
  3. 查找
     1. 查找表的相关概念；
     2. 顺序表的查找：算法与分析；
     3. 有序表的查找：折半查找，判定树及其查找分析等；
     4. 索引顺序表的查找：算法与分析；
     5. 动态查找表的类型定义：二叉排序树的构造、插入、删除等算法；
     6. 平衡二叉树；
     7. B-树和B+树；
     8. 哈希表：哈希函数与、处理冲突的方法，查找分析。
  4. 内部排序
     1. 排序的定义；
     2. 排序方法的分类；
     3. 插入排序；
     4. 快速排序；
     5. 选择排序；
     6. 归并排序；
     7. 基数排序；
     8. 各种内部排序方法的比较。
  5. 程序设计
     1. 顺序结构程序设计；
     2. 选择结构程序设计；
     3. 循环控制；
     4. 指针、动态内存分配；
     5. 数组、结构体；
     6. 函数、类型定义等基本预编译命令。

## 四、掌握重点

1. 算法的时间复杂度和空间复杂度分析；
2. 线性表、栈、队列、串、数组的表示、基本算法实现和应用；
3. 串的模式匹配算法。
4. 矩阵的压缩存储；
5. 广义表的存储和递归算法；
6. 二叉树的定义及存储结构；
7. 遍历二叉树和线索二叉树；
8. 树的各种存储结构，树、森林与二叉树的转换，树和森林的遍历；
9. 哈夫曼树与哈夫曼编码；
10. 回溯法与树的遍历，树的计数。
11. 图的邻接矩阵和邻接表存储表示和图的BFS和DFS遍历；
12. 拓扑排序、关键路径、最短路径；
13. 顺序表顺序查找算法与分析、有序表的折半查找与查找算法分析；
14. 二叉排序树、平衡二叉树、B-树和B+树；
15. 哈希表的构造与查找；
16. 插入排序、快速排序、选择排序、归并排序、基数排序；
17. 规范的程序设计。

# 考试内容——笔记

## 数据结构概述

### 数据结构的基本概念以及相关术语

**数据：**数据是信息的载体，是描述客观事物属性的数、字符及所有能输入到计算机中并被计算机程序识别和处理的符号的集合。数据是计算机程序加工的原料。

**数据元素：**数据的基本单位，通常作为一个整体进行考虑和处理。

**数据项：**构成数据元素的不可分割的最小单位，一个数据元素由若干个数据项组成。

**数据对象：**具有相同性质的数据元素的集合，是数据的一个子集。

**数据结构：**相互之间存在一种或多种特定关系的数据元素的集合。

### 数据结构的抽象数据类型表示与实现

**数据类型、抽象数据类型**

* **数据类型：**一个值的集合和定义在此集合上的一组操作的总称
  + **原子类型：**其值不可再分的数据类型。
  + **、结构类型：**其值可以再分解为若干个分量的数据类型。
* **抽象数据类型（Abstract Data Type, ADT）是抽象数据组织及与之相关的操作**

**逻辑结构——数据之间的逻辑关系**

* 集合：各个元素同属于一个集合，别无其他关系
* 线性结构：数据元素之间一对一的关系；除了第一个元素，所有元素都有唯一前驱；除了最后一个元素，所有元素都有唯一后继。
* 树形结构：数据元素之间是一对多的关系。
* 图结构：数据元素之间是多对多的关系

**数据的运算——针对某种逻辑结构，结合实际需求，定义基本运算**

* **线性结构**
  + 查找第i个数据元素
  + 在第i个位置插入新的数据元素
  + 删除第i个位置的数据元素

**数据结构的物理结构（存储结构）**

* **线性结构**
  + **顺序存储：**把逻辑上相邻的元素存储在物理位置上也相邻的存储单元中，元素之间的关系由存储单元的邻接关系来体现。
  + **链式存储：**逻辑上相邻的元素在物理位置上可以不相邻，借助指示元素存储地址的指针来表示元素之间的逻辑关系。
  + **索引存储：**在存储运算信息的同时，还建立附加的索引表。索引表中的每项称为索引项，索引项的一般形式是（关键字，地址）
  + **散列存储：**根据元素的关键字直接计算出该元素的存储地址

### 算法的基本概念及设计要求

**程序 = 数据结构 + 算法**

**算法：**是指对特定问题求解步骤的一种描述，是指令的有限序列，其中每条指令表示一个或多个操作。

**算法特性**

* **有穷性：**一个算法必须总在执行有穷步之后结束，且每一步都可在有穷时间内完成。
* **确定性：**每条指令必须有确切的含义，对于相同的输入只能得出相同的输出。
* **可行性：**描述的操作都可以通过已经实现的基本运算执行有限次来实现。
* **输入和输出：**有一个或多个输入和输出

### 算法的时间复杂度和空间复杂度分析

**时间复杂度**

**O(1) < O() < O(n) < O() < O() < O() < O< O(n!) < O()**

**平均时间复杂度：**所有输入示例等概率出现的情况下，算法的期待运行时间。

**最坏时间复杂度：**最坏情况下算法的时间复杂度。

**空间复杂度**

**算法原地工作——算法所需内存空间为常量**

## 线性表

### 线性表的类型定义、基本算法，以及集合运算等应用问题的解决方案；

**线性表：**具有相同数据类型的n(n>=0) 个数据元素的有限序列，其中n为表长，当n=0时线性表是一个空表。若用L命名线性表，则其一般表示为：；

**直接前驱：**除第一个元素外，每个元素有且仅有一个直接前驱；

**直接后继：**除最后一个元素外，每个元素有且仅有一个直接后继；

**线性表的基本操作**

* **InitList(&L):**初始化表。构造一个空的线性表L，分配内存空间
* **DestroyList(&L)：**销毁操作，销毁线性表，并释放线性表L所占用的内存空间
* **ListInsert(&L, i, &e)：**插入操作，在表L中第i个位置上插入指定元素e**。**
* **ListDelete(&L, i, &e)：**删除操作，删除表L中第i个位置的元素，并用e返回删除元素的值。
* **LocateElem(L, e)：**按值查找，在表L中查找具有给定关键字值的元素。
* **GetElem(L, i)：**按位查找，获取表L中第i个位置的元素的值。
* **Length(L)：**求表长，返回线性表L的长度，即L中数据元素的个数
* **PrintList(L)：**输出操作，按照请后顺序输出线性表L的所有元素值
* **Empty(L)：**判空操作，判断线性表L是否为空。

### 线性表的顺序表示和实现，及集合运算等应用问题的解决方案的顺序表实现；

**顺序表：**用顺序存储的方式实现线性表

**顺序存储：**把逻辑上相邻的元素存储在物理位置上也相邻的存储单元中，元素之间的关系由存储单元的邻接关系来体现。

**顺序表特点：**

* **随机访问：**在O(1)时间内找到第i个元素
* **存储密度高**
* **拓展容量不方便**
* **插入、删除不方便，需要移动元素多**

**静态分配：**

*// 静态分配*

*#define* MaxSize 10 *// 定义最大长度*

typedef struct{

    ElemType data[MaxSize]; *// 用静态“数组”存放数据元素*

    int length; *// 顺序表当前的长度*

}SqList; *// 顺序表类型定义*

* **插入操作ListInsert(SqList &L, int i, int e)**

*// 插入*

*#define* MaxSize 20

typedef struct{

    int data[MaxSize];

    int length;

}SqList;

*// 初始化顺序表（静态分配）*

void InitList(SqList &L){

    L.length = 0;

}

*// 在顺序表中第i处插入元素e*

bool ListInset(SqList &L, int i, int e){

*if*(i <= 1 || i >= L.lengtg + 1)

*return* false;

*if*(L.length >= MaxSize)

*return* false;

*for*(int j = L.length; j >= i; j--){

        L.data[j] = L.data[j - 1];

    }

    L.data[i - 1] = e;

    L.length++;

*return* true;

}

int main(){

    SqList L;

    InitList(L);

*/// 代码, 顺序表中插入元素*

    ListInsert(L, 3, 3);

*return* 0;

}

* **删除操作：**

*// 删除*

bool ListDelete(SqList &L, int i, int &e){

*if*(i < 1 || i > L.length)

*return* false;

    e = L.data[i - 1];

*for*(int j = i; j < L.length; j++){

        L.data[j-1] = L.data[j];

    }

    L.length--;

*return* true;

}

* **按位查找：**

*// 按位查找*

ElemType GetElem(SqList L, int i){

*return* L.data[i-1];

}

* **按值查找：**

*// 按值查找*

int LocateElem(SqList L, ElemType e){

*for*(int i = 0 ; i < L.length; i++){

*if*(L.data[i] == e){

*return* i + 1;

        }

    }

*return* 0;

}

**动态分配：**

*// 动态分配*

*#define* InitSize 10 *// 顺序表的初始长度*

typedef struct{

    Elemtype \*data; *// 指示动态分配数组的指针*

    int MaxSize; *// 顺序表的最大容量*

    int length; *// 顺序表的当前长度*

}SeqList; *// 顺序表的类型定义（动态分配方式）*

*// 顺序表初始化（动态分配）*

void InitList(SeqList &L){

    L.data = (ElemType \*)malloc(sizeof(ElemType) \* InitSize);

    L.length = 0;

    L.MaxSize = InitSize;

}

*// 增加动态数组长度*

void IncreaseSize(SeqList &L, int len){

    ElemType \*p = L.data;

    L.data = (ElemType \*)malloc(sizeof(ELemType) \* (L.MaxSize + len));

*for*(int i = 0; i < L.length; i++){

        L.data[i] = p[i];

    }

    L.MaxSize = L.MaxSize + len;

    free(p);

}

*// 动态申请和释放内存空间*

*// C——申请 malloc*

L.data = (Elemtype \*)malloc(sizeof(ElemType) \* InitSize)

*// C++ —— 申请*

L.data = new ELemType[InitSize]

*// 释放 free\delete*

### 线性表的链式表示和实现，及集合运算等应用问题的解决方案的单链表实现；

### 一元多项式的表示及相加，即一元多项式的单链表加法算法实现。