# 数据结构与算法分析笔记

YyyXxxLll

## 目录



## 1 如何看懂一个程序？

1. 分析流程
2. 分析每一条语句的功能
3. 试数

# 2 前置知识

## 2.1 线性结构

所有的结点可以用一根直线穿起来

**连续存储** 数组

**离散存储** 链表

### 2.1.1 线性结构的应用

1. 栈
2. 队列

## 2.2 离散存储

## 2.3 非线性结构

树 + 图

## 2.4 查找和排序

折半查找

冒泡排序、快排

# 3 什么是数据结构？

我们如何把现实生活中大量而复杂的问题以某种特定的数据类型（数据以什么方式存储）和特定的存储结构（数据个体间的关系）保存到**内存**中。

数据结构 = 个体 + 个体的关系

# 4 算法

在数据结构的基础上为实现某个功能（查找、删除、排序 ……）而对数据进行的操作叫做算法。

算法 = 对被存储的数据操作

从狭义上讲，存储的方式不一样，要使用的算法也不一样，例如：遍历一个数组很容易，但遍历一个链表很难。

从广义上讲，算法和数据的存储方式无关（泛型）。

## 4.1 衡量算法的标准

时间复杂度、空间复杂度、难易程度、健壮性

### 4.1.1 时间复杂度

程序的大概执行次数，并非执行时间（因为机器和机器之间的速度不同）

### 4.1.2 空间复杂度

程序执行过程中大概占用的最大内存

#### 4.1.3 难易程度

#### 4.1.4 健壮性

# 5 预备知识

1. 指针
2. 结构体
3. 动态内存的分配和释放

## 5.1 结构体

用户根据自己的需要定义的复合数据类型

#### 5.1.1 为什么需要结构体？

为了表示复杂的数据类型，而不同的数据类型无法满足需求

#### 5.1.2 如何定义一个结构体？

图1 定义了一个结构体。

#### 5.1.3 使用结构体的注意事项

1. 用结构体定义一个数据对象需要带`struct`。
2. 用结构体定义的对象不能进行加减乘除，但可以互相赋值。如图2 所示（注意：结构体还涉及字节对齐的知识。）：

## 5.2 动态内存的分配和释放

图3 定义了一个动态数组，`malloc(...)`前面的`(int *)`的含义：`malloc`默认返回的是这段内存第一个字节的地址，加上`(int *)`（一个强制类型转换）是为了告诉这个地址是整形数据的地址，这样一来`ptr[0]`表示的是第 0~3 个的数据，`ptr[1]`表示的是第 4~7 个内存单元的数据（待勘误）。在图3 的例子中，我们可以通过需求动态地构造一个数据（`len`），还可以手动释放这个数组空间。

#### 5.2.1 跨函数使用内存

图 4 通过返回一个动态内存的地址来实现跨函数使用内存。

# 6 链表

$N$ 个结点离散分配，每个结点通过指针相连，每个结点只有一个前驱（首结点无）、一个后继（尾结点无）。

## 6.1 一些术语

首结点、尾结点、头结点、头指针

如果需要通过一个函数来对链表进行处理，我们最重要的接收到链表的头指针，通过头指针可以推导出链表的其他信息。

```
    #define NAME    20
    struct Student
    {
        char name[NAME];
        int age;
        int id;
    };
    int main(void)
    {
        struct Student A = {...};
        return 0;
    }
/---------------另一种方法---------------------------------/

    #define NAME    20

    #typedef struct
    {
        char name[NAME];
        int age;
        int id;
    } Student;

    int main(void)
    {
        Student A = {...};

        return 0;
    }
```

图 1: 定义一个结构体

## 6.2 链表分类

1. 单链表
2. 双链表
3. 循环链表
4. 非循环链表

## 6.3 链表涉及的算法

1. 遍历
2. 插入
3. 删除
4. 清空

```
struct Student A = {...};
struct Student B = {...};
struct C = A + B;    // ERROR!!!
struct C = A;        // RIGHT!!!
```

图 2: 结构体的使用

```
#include <malloc.h>
    ...
int len;
scanf("%d", &len);
(int *)ptr = (int *)malloc(sizeof(int) * len)
ptr[0] = 10;
...
free(ptr)
```

图 3: 定义一个动态数组

5. 排序
6. 查找

#### 6.3.1 链表的算法的实现

```c
#include <stdio.h>
#include <malloc.h>

typedef int Age;
typedef struct
{
    char name[40];
    Age age;
}Student;

Student * creatInfoTable(void);

int main(void)
{
    Student * ptr = creatInfoTable();
    printf("%d\n", ptr->age);
    return 0;
}

Student * creatInfoTable(void)
{
    Student * ptr_tab = (Student *)malloc(sizeof(Student));
    ptr_tab->age = 100;
    return ptr_tab;
}
```

图 4: 跨函数使用内存