数据结构01

概述

数据结构定义

如果把现实中大量复杂的问题以**特定的数据类型**和**特定的存储结构**保存到主存储器，在此基础上，为实现某个功能（如，查找某个元素，删除某个元素，对所有元素进行排序）而执行的相应操作，这个操作也叫算法

数据结构 = 个体存储 + 个体关系存储

算法 = 对存储数据的操作

狭义的算法——》与数据的存储方式密切相关

广义的算法——》与数据的存储方式无关

泛型：利用某种技术达到，不同的存储方式，执行的操作是一样的效果

算法：解题的方法和步骤

衡量算法的标准

1.时间复杂度：程序要执行的次数，而不是执行的时间

2.空间复杂度：执行过程中大概占用的最大内存

3.可读性

4.耐造性

预备知识

指针

定义：构建一个变量来储存其他变量的地址（内存单元的编号0–(4G–1)）

指针就是地址///指针变量是存放内存单元地址的变量

指针本质是一个操作受限（只能相减）的非负整数

e.g.

int *p // p 为指针变量名字，int * 表示p只能储存int类型变量的地址

int a = 5;

p = &a;

printf (a, &a, p, &p)； ————》》5，f4, f4, f7

指针与函数调用

```
# include <stdio.h>

void f(int * p) //不是定义了一个名字叫做*p的形参，而是定义了一个形参，
{
    *p = 100;
}

int main(void)
{
    int i = 9;

    f(&i);
    printf("i = %d\n", i);

    return 0;
}
```

i = 100;

指针与数组

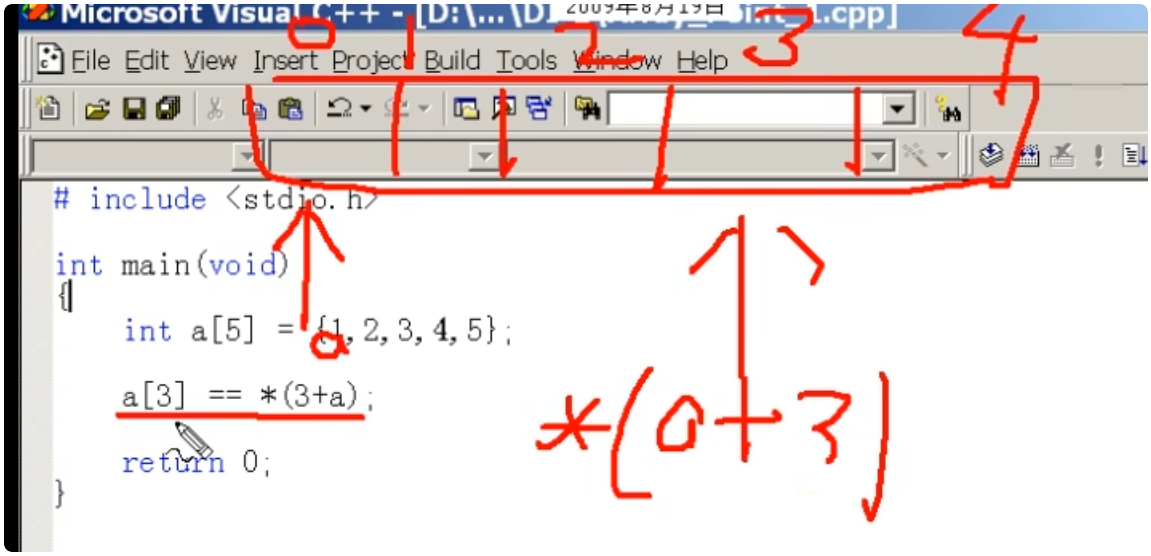
一维数组名是个指针常量

which 存放的是数组中第一个元素的地址——》指向第一个元素

其值不能改变

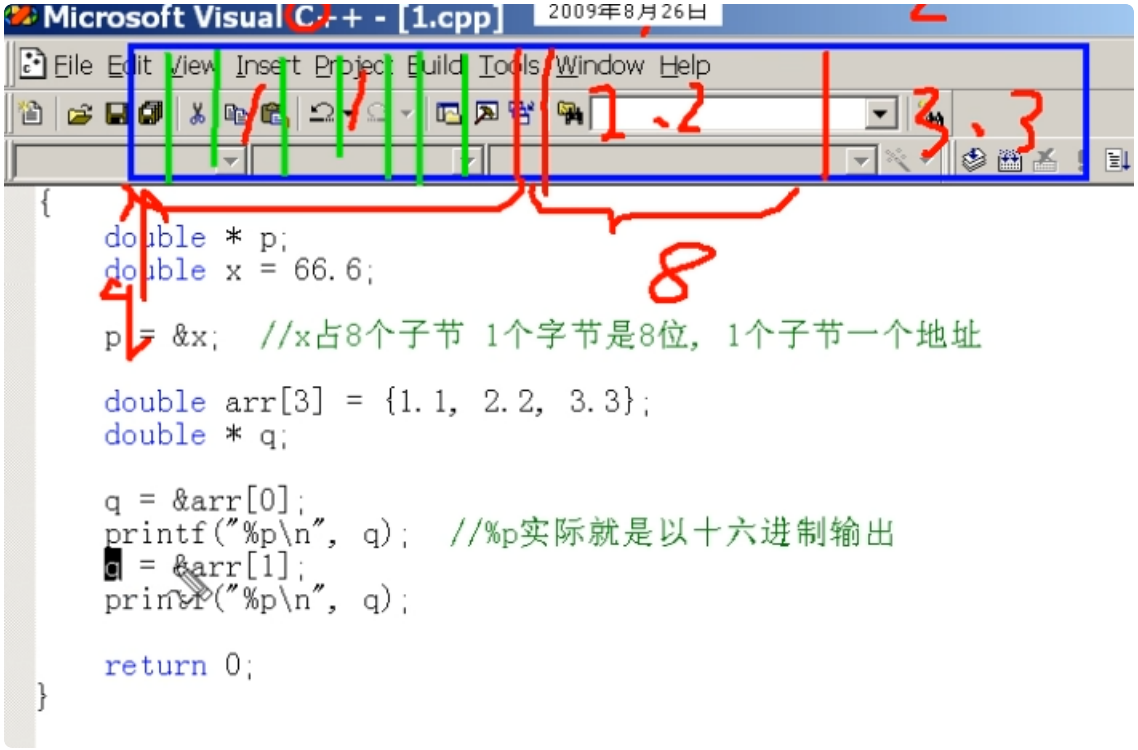
a 是数组里第一个数的地址

a[3] == *(a + 3)



指针的大小

数组指针只指向第一个元素的首部（如图）



故，指向200个字节和指向2个字节的指针大小相同

不管改什么变量，只要改所指地址就可以改变其值（**不懂**

```c
int main(void)
{
    int i = 9;
    int * p = &i;// int  *p;  p = &i;

    printf("%p\n", p);
    f(&p);
    printf("%p\n", p);

    return 0;
}

void f(int ** q)
{
    *q = (int *)0xFFFFFFFF;
}
```

结构体

为什么要用结构体?

为了表达一些复杂的数据，而普通的基本变量类型无法满足要求

把一个事物的多个变量放到一起表达，避免重复（如，学生的学号、班级。姓名）

什么叫结构体?

用户根据自己需要定义的复合数据类型

和类相比，结构体没有方法

类

```c
class Studnet
{
    int sid;
    String name;
    int sage;

    void inputStudent()
    {
    }

    void showStudent()
    {
    }
}
```

结构体

类型：struct Studnet

成员：int sid;

String name;

int age;

```
struct Studnet
{
    int sid;
    String name;
    int sage;
}
```

使用结构体创造新变量的两者写法

```
两种方式：
    struct Student st = {1000, "zhangsan", 20};
    struct Student * pst = &st;

    1.
        st.sid
    2.
        pst->sid
```

```
int main(void)
{
    struct Student st = {1000, "zhangsan", 20};
    printf("%d  %s  %d\n", st.sid, st.name, st.age);

    st.sid = 99;
    //st.name = "lisi";  //error
    strcpy(st.name, "lisi");
    st.age = 22;
    printf("%d  %s  %d\n", st.sid, st.name, st.age);

    return 0;
}
```

注意事项

结构体变量不能加减乘除，但可以相互赋值

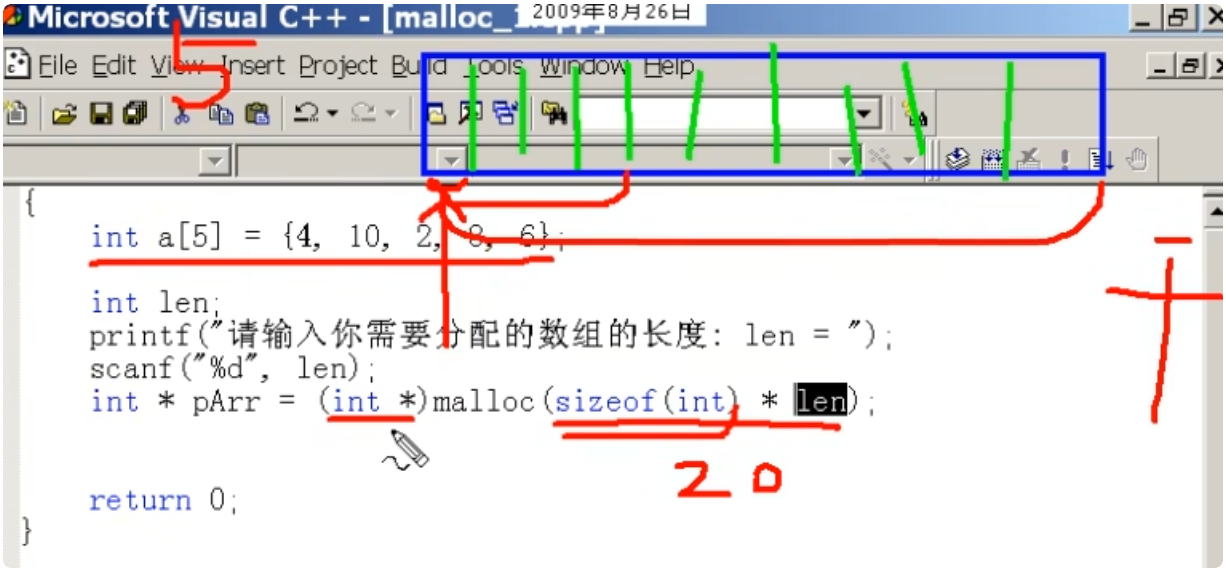
普通结构体变量和结构体指针变量作为函数传参的问题

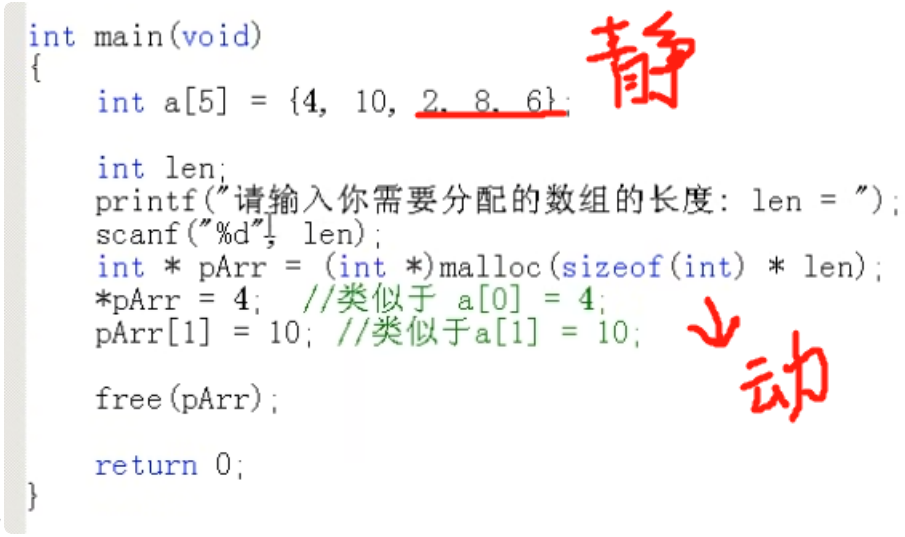
动态内存的分配和释放

什么是静态，什么是动态的?

使用函数malloc（）——》动态

（int *）的含义：由于malloc只返回第一个字节的地址，我们不知道该数据是什么类型的（是占4个还是8个字节的数据），所以在前面强制转换为int，告诉编译器这是int数据的第一个字节的地址





静态动态数组的对比

线性结构：连续存储【数组】

线性结构：把所有的结点用一根直线穿起来

数组 ：元素类型相同，大小相等

优点

存取速度很快

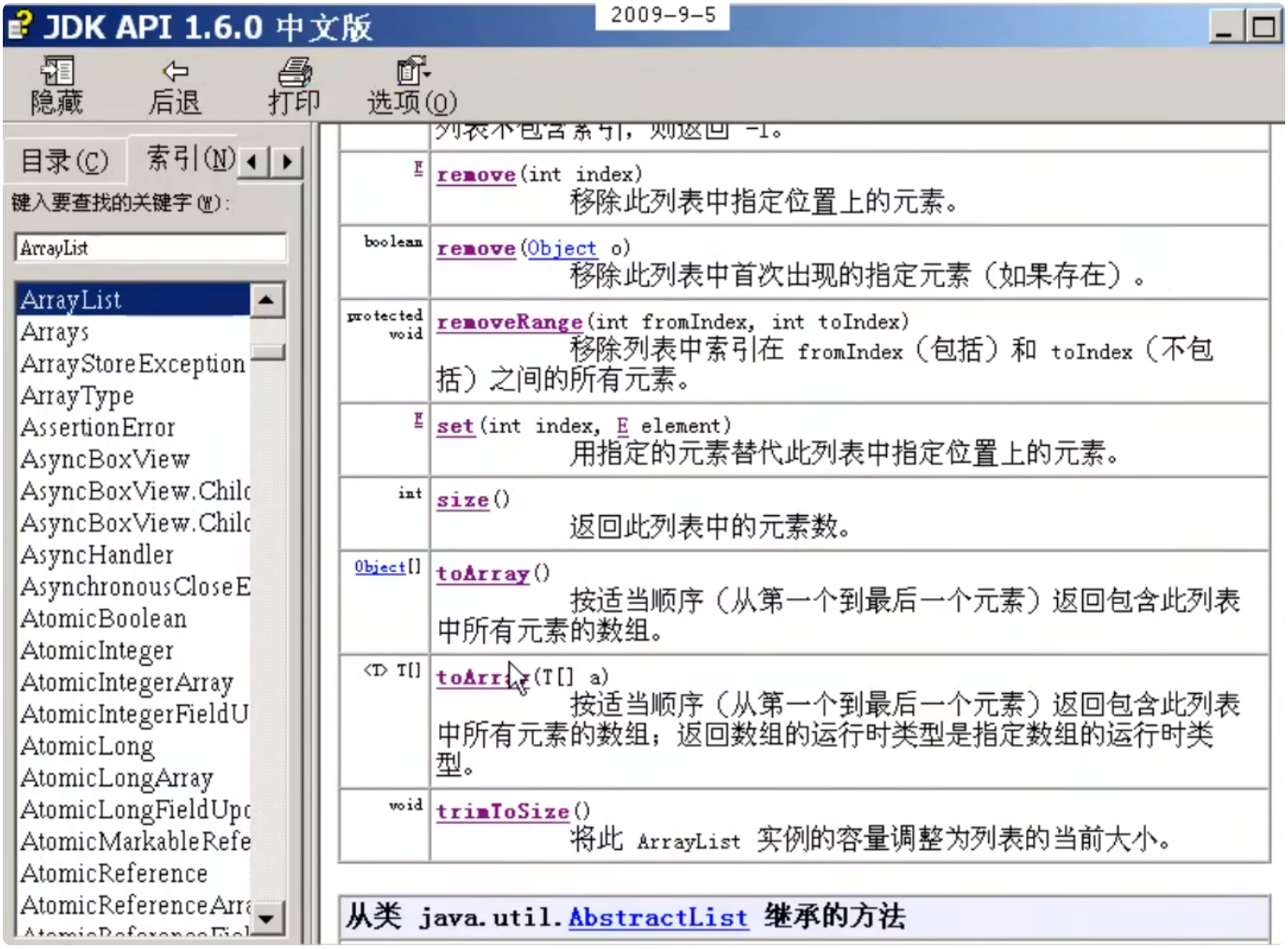
缺点

插入和删除元素很慢

空间通常是有限的

如何创造储存数组的结构体

看看官方使用文档



定义结构体

定义了一个数据类型（struct Arr），该数据类型含有三个成员（pBase, len, cnt, increment）

```
struct Arr
{
    int * pBase; //存储的是数组第一个元素的地址
    int len; //数组所能容纳的最大元素的个数
    int cnt; //当前数组有效元素的个数
    int increament; //自动增长因子
};
```

增加功能（函数）

```
void init_arr();
bool append_arr(); //追加
bool insert_arr();
bool delete_arr();
int get();
bool is_empty();
bool is_full();
void sort_arr();
void show_arr();
void inversion_arr();
```

构建函数（重点，看视频）

不懂为什么给array赋值不能赋到arr 26.38

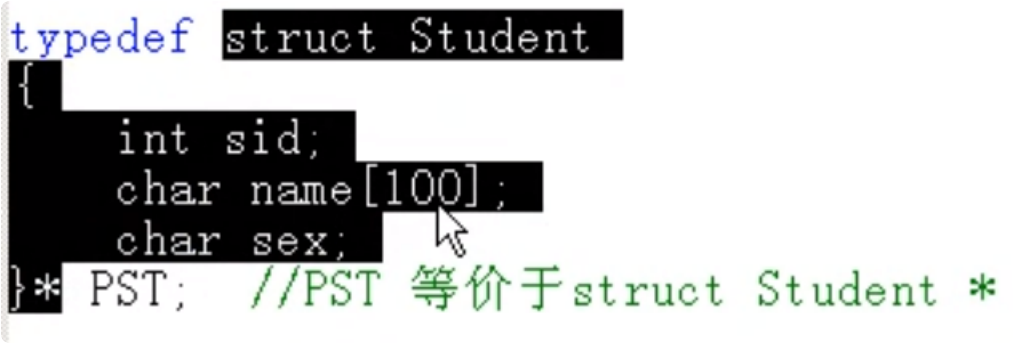
```
int main(void)
{
    struct Arr arr;

    init_arr(arr);

    return 0;
}

void init_arr(struct Arr array)
{
    array.len = 99;
}
```

typedef

为已用的数据类型再多取一个名字——》当我们建立结构体时，不用这么麻烦打很多字（struct student st）

typedef int ZHANGSAN // 为int多取一个名字 int == ZHANGSAN, int i ==ZAHNGSAN

拓展 给struct Student* 这个指针变量起个名字




```
typedef struct Student
{
    int sid;
    char name[100];
    char sex;
}* PSTU, STU; //等价于ST代表了struct Student,  PST代表了struct Student *
```

线性结构：离散存储【链表】

定义

n个节点离散分配

彼此通过指针相连

每个节点只有一个前驱节点和一个后续节点

优点

空间没有限制

插入和删除元素很快

缺点

存取速度很慢

专业术语

首节点：第一个有效节点

尾节点：最后一个有效节点

头节点：

首节点前面的节点

不存放有效数据

和后面节点的数据类型一样

目的是为了方便对链表的操作

头指针：指向头节点的指针变量

尾指针：指向尾节点的指针变量

数据域：节点存放数据的部分

指针域：节点存放下一个节点整体的地址的部分

确定一个链表的必要参数——》头指针即可

指针的分类

单链表

双链表：每一个节点有两个指针域（指向前后指针）

循环链表：能通过任何一个节点找到其他所有节点

非循环链表

算法：

遍历

查找

清空

销毁

求长度

```
typedef struct node
{
    int data;    //存储数据本身
    struct node *pNext;    //pNext指向一个和它本身存储指向下一个结点的指针
}NODE, *PNODE;
```

- 解释：
- NODE等价于struct node
- PNODE等价于struct node *

```
PNODE p = (PNODE)malloc(sizeof(NODE));
```

- 解释：
- 将动态分配的新节点的地址赋给p

```
free p;    //删除p指向结点所占的内存，不是删除p本身所占内存
p->pNext;    //p所指向结构体变量中的pNext成员本身
```